1-1-2017

Improving Usability And Scalability Of Big Data Workflows In The Cloud

Aravind Mohan
Wayne State University,

IMPROVING USABILITY AND SCALABILITY OF BIG DATA WORKFLOWS IN THE CLOUD

by

ARAVIND MOHAN

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2017

MAJOR: COMPUTER ENGINEERING

Approved By:

_____

Advisor Date

_____

Co-advisor

_____

_____

_____

DEDICATION

To God be the glory.

ACKNOWLEDGEMENTS

I would first like to thank God for giving me an amazing opportunity to pursue a Ph.D. degree in a fascinating field of Computer Engineering, for giving me a passion to research, for bringing me to Wayne State University to work with my advisors Dr. Shiyong Lu and Dr. Song Jiang, for guiding me through a number of important decisions, for protecting me, for answering my prayers, and for giving me the strength to persevere. I would also like to express my deep and sincere gratitude to my advisor Dr. Shiyong Lu, for his guidance, encouragement, and support throughout my Ph.D. studies. Dr. Lu's advice has enabled me to remain focused and to succeed in my studies. I am deeply thankful for Dr. Lu's kindness and for inspiring me to pursue an academic career. I am also very grateful to my dissertation committee members: Dr. Shiyong Lu, Dr. Jiang Song, Dr. Alexander Kotov, Dr. Caisheng Wang, and Dr. Fengwei Zhang, for being on my dissertation committee and for providing their helpful feedback, insightful comments, and constructive suggestions.

I would also like to thank Dr. Fotouhi for his encouragement and support during my Ph.D. studies. I am deeply thankful for the University Graduate Research Assistant and Graduate Teaching Assistant, and Part Time Faculty positions that has enabled me to have an early start in my Ph.D. research and to have a very strong teaching experience. I would also like to thank the Budget Office department at Wayne State University for providing me the Graduate Student Assistant position that has helped me to fund my education in the first two years of my doctoral studies and for providing me an opportunity to exercise my programming skills. I would also like to thank my bright colleagues from the Big Data Research Laboratory: Mahdi Ebrahimi, Dong Ruan, Fahima Bhuyan, Ishtiaq Ahmed, and Changxin Bai, as well as alumni Dr. Andrey Kashlev, Dr. Cui Lin, and Dr. Xubo Fei. I would like to express my deep appreciation to Mahdi Ebrahimi for the strong collaboration and friendship that has resulted in several research publications and in implementing the DATAVIEW system.

I am especially thankful to my wife and my entire family, who has been incredibly supportive throughout my studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

In recent years, the term "big data" has become a buzzword in both industry and academia. It is frequently used to refer to the method of processing and analyzing large amount of heterogeneous types of data to mine hidden patterns, correlations, trends, and other useful information to guide enterprise decisions and strategies for cutting cost or increasing revenues [1, 2].

Scientific workflows, which were originally developed for automating the data analysis process of scientific data (often large in volume and vary in types) to enable and accelerate scientific discovery, are now increasingly used for the processing and analysis of business data as well. Big data workflows have recently emerged as the next generation of data-centric workflow technologies to address the five "V" challenges of big data: volume, variety, velocity, veracity, and value. More formally, a big data workflow is the computerized modeling and automation of a process consisting of a set of computational tasks and their data interdependencies to process and analyze data of ever increasing in scale, complexity, and rate of acquisition. The convergence of big data and workflows creates new challenges in workflow community.

First, the variety of big data results in a need for integrating large number of remote Web services and other heterogeneous task components that can consume and produce data in various formats and models into a uniform and interoperable workflow. The shimming problem has received much attention in the workflow community [3, 4, 5, 6], in which special kinds of adaptors, called shims, are inserted between workflow tasks to resolve the data type incompatibility issue. However, to meet the challenges of big data, several issues in the existing approaches need to be examined. First issue is to scale the registration and configuration procedure to large number of workflow tasks, which is critical for large workflows. Second issue is to ease the integration of large number of remote Web services and other heterogeneous task components such as command line executables that can consume and produce data in various formats and data models into a uniform and interoperable workflow. In particular, we examine automatic shim generation techniques for third-party Web services, whose development were not workflow-aware. Existing approaches provide solutions to the shimming problem only through adhoc mechanism and fall short in providing generic solution. We address this limitation by automatically inserting a piece of code called shims or adaptors in order to resolve the data type mismatches.

Second, the volume of big data results in a large number of datasets that needs to be queried and analyzed in an effective and personalized manner. Further, there is also a strong need for sharing, reusing, and repurposing existing tasks and workflows across different users and institutes. However, existing workflow systems are mainly single-user oriented with limited sharing and reusing functionalities. To overcome such limitations, we propose a folksonomy-based social workflow recommendation system to improve workflow design productivity and efficient dataset querying and analyzing.

Third, the volume of big data results in the need to process and analyze data of ever increasing in scale, complexity, and rate of acquisition. But a scalable distributed data model is still missing that abstracts and automates data distribution, parallelism, and scalable processing. In the meanwhile, although NoSQL has emerged as a new category of data models, they are optimized for storing and querying of large datasets, not for adhoc data analysis where data placement and data movement are necessary for optimized workflow execution. We propose a NoSQL collectional data model that addresses this limitation.

Finally, the volume of big data combined with the unbound resource leasing capability foreseen in the cloud, facilitates data scientists to wring actionable insights from the data in a time and cost efficient manner. We propose the BARENTS scheduler that supports high-performance workflow scheduling in a heterogeneous cloud computing environment with a single objective to minimize the workflow makespan under a provided budget constraint.

The dissertation is organized as follows: Chapter 2 reviews the research background that covers the shimming problem and the primitive workflow model, the folksonomy based social recommendation system for big data workflows, the NoSQL collectional data model, and the BARENTS scheduler. Chapter 3 presents our work on the shimming problem and the primitive workflow model. Chapter 4 presents our work on a folksonomy based social recommendation system for big data workflows. Chapter 5 presents our work on our proposed NoSQL Collectional Data Model. Chapter 6 presents our scheduling algorithm called BARENTS scheduler, which is used to schedule Big Data Workflows in the Cloud under a user defined budget constraint. Finally, Chapter 7 concludes the dissertation by proposing future work and possible research directions.

CHAPTER 2: RELATED WORK

2.1 The Shimming Problem and Primitive Workflow Model

The notion of the shimming problem was first coined in [6]. Due to the heterogeneous nature of the data involved in solving complex scientific problems, output of one program or application will not be always compatible with another program or application and hence the interoperability of both programs and applications to work together is a challenge. The challenge of interoperability between the mixed data types is often termed as "shim" and there has been much work done in classifying and solving the shimming problem. Shim is broadly classified into 1) semantic compatibility 2) syntactic compatibility. In [6], the author introduced shims as a technique to align or mediate mismatching third party Web services that has incompatible input and output. Other work has been done to solve the shimming problem in literature such as [11] in which a mapping language is proposed to use OWL ontologies in order to capture the semantics of a data source and the instances of these ontology concepts are used to support conversion of data between different formats. Although this technique introduces the ontology model that can be used to solve the shimming problem to some extent from the semantic incompatibility perspective, it is very complex to implement and is not usable. In [1], the author proposed an approach to solve the shimming problem by automatically inserting the shims into scientific workflow but the approach only supports relational data and does not have any contribution towards other data types.

In [12], the author proposed the task model in scientific workflow and classified the shimming problem into TYPE-I and TYPE-II shimming problems. Further this approach uniquely allows shims to be either invisible or visible at the workflow level, supporting both functional and operational perspectives of scientific workflows Although the work was well cited and accepted in the scientific workflow community, the proposed task registration model is much complicated as the model contains more than one component within a single task and the shimming needs to be done while registering a task into the system through complex mapping between the components within a task, therefore complicates the system and is not usable. In [13], the author proposed to reduce the shimming problem in scientific workflow to the runtime coercion problem and defined the notion of well typed workflow. But this approach did not address the challenges incurred during registration of a p-workflow and did not answer the question on how the mapping occurs

from the input/output port of the p-workflow to the input/output of the p-workflow component and did not provide any solution to guarantee interoperability between the p-workflow and other upstream/downstream workflows connected to it .

None of the above technique addresses the shimming problem that exist in registering third party applications such as web services, command lines applications as a p-workflow , which is the focus of this paper. In this paper, we provide two solutions:

1. For a command line based application, we have formulated the IP2I and O2OP mapping table and proposed the solution in a case by case approach.

2. For a Web service based application, we have provided a solution to separate the registration from the configuration, thereby easing the registration process of a Web service operation as a p-workflow into our DATAVIEW system. Shimming is automatically done inside the p-workflow using the user driven configuration mechanism. In addition, our approach also parses the WSDL file automatically to get the input and output information and hence automates the conversion of a workflow unaware Web service operation as a p-workflow from end to end and manage to guarantee interoperability with both upstream and downstream workflows.

We also addressed challenges incurred due to the large datasets that needs to be stored and managed in scientific workflow, by integrating MongoDB , a NOSQL database within our DATAVIEW system as a preliminary step to do big data analyzing and processing. Our data product manager provides a layer of abstraction on top of the MongoDB and thereby not making major changes to our DATAVIEW architecture [7].

2.2 A Folksonomy Based Social Recommender System for Workflow and Data Reuse

The notion of artifact reuse and recommendation is well studied in the software engineering field. The folksonomy based approach, in which the system provides the users with the ability to publish and categorize various resources such as (web pages, photos, videos, documents, etc.) online with “social annotations” or “tags”. In [16], the author explore three properties of folksonomy, namely the categorization, the keyword and the structure property and propose a personalized search framework utilizing the folksonomy. In the information retrieval community, the author [17] propose

a formal model and a new search algorithm for folksonomies. In [18], the author explore the social annotations to optimize web search. In the paper, the author propose two novel algorithms called SocialSimRank and SocialPageRank to measure the similarity and popularity of web pages from web users perspective.

In the scientific workflow community, building visualization and workflow pipelines is a large hurdle from the users' perspective. It is time consuming to identify a reusable workflow by manually scanning through more than hundreds of workflows in the workflow repository. In [19], the authro propose VisComplete, an autocomplete suggestion technique to help users construct pipelines in the VisTrails system. In the paper, the authors propose a technique to find the syntactic similarity between the incomplete workflow and the workflows in the workflow repository by sub graph matching. In [20], the author propose a case-based reasoning approach to assist composition of workflows using the Lesk algorithm to perform the keyword matching between the input and output of the incomplete workflow and the workflows in the workflow repository. In [21], the author propose an approach to recommend services in the workflow composition process. The author models existing scientific artifacts, services and workflows as a PSW network and recommends services based on the service usage history.

None of the above techniques address the workflow reuse problem using the syntactic and semantic information available in the workflow specification file. Further the above techniques do not address the scientific workflow reuse in the granularity of recommending the producer and consumer workflows based on the input and output port type matching. Our technique validates the port types to recommend syntactically compatible workflows. Next, we leverage the user profile and the tags associated with the incomplete workflow and the workflows residing in the workflow repository to recommend a suitable workflow that is both relevant and preferred by the user to be a producer or consumer of the incomplete workflow.

2.3 A NoSQL Collectional Data Model

Existing workflow management systems such as [33–35] does not support a scalable data model that is suitable for processing big data in the cloud. Kepler proposes a collection oriented model in which the data is nested in different levels as collections and sub collections with arbitrary data type. The data model is represented using XML and is semi structured in nature, whereas our

collectional data model is structured and is much simple to process big data workflows. VisTrails provide a good visualization framework and support a semi structured representation of the data, it strongly lacks in storing hierarchical information such as collections and the workflow engine does not support scalable workflow execution in the cloud. Taverna supports singleton values such as strings, byte arrays and list of singletons. Lists are defined to a specific level and are not capable to handle nested data to arbitrary levels.

In [30,32], the author propose a collectional data model to process scientific workflows in one machine and a set of well-defined operators and constructs. The proposed model and the operators are not scalable and do not consider the data partitioning and the workflow execution in the cloud. In [31], the author propose an approach to improve the programmability and scaling flexibility of big data application through different parallelization techniques. They propose a list of DDP patterns that are used to process key value pairs and parallelize the execution of the user-defined functions. Although their approach is similar to the workflow constructs proposed by us, our Map and Reduce workflow constructs can be applied to any given workflow by leveraging our proposed NoSQL collectional data model.

Existing big data NOSQL databases are classified into the following four category of databases: 1) key-value databases [36], such as Memcached and Redis, 2) document-oriented databases [37], such as RavenDB, MongoDB and CouchDB, 3) column-family databases [38, 40], such as Apache Cassandra and HBase, 4) graph databases [39], for example: Neo4J, FlockDB and GraphDB. The existing NOSQL databases are not suitable for big data workflows because they do not support ad hoc sophisticated analysis and is not extendible to workflow optimization.

None of the above techniques provides a scalable data model for data centric big data workflows. In this paper, we propose a NoSQL collectional data model that is both scalable and at the same time is well structured for performing ad hoc analysis on large datasets. Further, we also propose two new cloud workflow executors that take advantage of the proposed NoSQL data model to improve the performance of workflow execution.

2.4 Scheduling Big Data Workflows in the Cloud under Budget Constraints

While the cloud computing paradigm [67–69] provides a promising platform for running big data workflows, performance tuning of workflow execution in the cloud remains an important and challenging problem. One challenge is the selection of cloud resources. Given a workflow w, how many virtual machines are needed to run w in the cloud? What types of virtual machines are needed? As a cloud typically provides a set of heterogeneous virtual machine types that come with different configurations and prices, the selection of such cloud resources often need to consider the characteristics of input datasets and workflows, and the QoS parameters provided by a user such as budget and deadline.

Over the past decade, there have been several workflow scheduling algorithms [74, 75], proposed that play a crucial role in running workflows efficiently on the cloud. The scheduling algorithms are widely categorized into static and dynamic algorithms. The existing static algorithms [51–54] do not consider the runtime estimation and hence are not very efficient in a heterogeneous cloud computing environment, where there is a cost and time for the computation performed in different resource types and a cost and time for the data movement from one resource to another. On the other hand, many existing dynamic algorithms [61, 62] are capable of adapting to the unexpected delays that occur while executing the workflow in the cloud.

The existing scheduling algorithms are either user driven with the QoS constraints set by the user or system driven with no constraints. There are two types of QoS constraints primarily proposed so far in the existing algorithms: 1) budget constraint [74, 75], 2) deadline constraint [72, 75]. Budget-constrained workflow scheduling algorithms aim to minimize the total execution time of a workflow while meeting a user specified budget constraint. Deadline-constrained workflow scheduling algorithms aim to minimize the total monetary cost of running a workflow while meeting a user specified deadline. There are some algorithms that belong to both categories [72, 75] as they aim to satisfy both the budget and deadline constraints, and hence are used for each category by relaxing one of the constraints. There are some algorithms that are solely based on system driven

optimization such as [57], which does not consider any constraints. The goal of those algorithms is to generate the schedule with a single objective, which is to minimize the makespan.

There have been several existing works in dynamic workflow scheduling algorithms. In [55], the author propose Dynamic Provisioning Dynamic Scheduling (DPDS) algorithm. The authors propose to schedule the workflow ensembles on the cloud by maximizing the execution of the total number of user prioritized workflows under a user provided QoS constraints such as budget and deadline. In [56], the author developed a probabilistic scheduling framework called Dyna that minimizes the execution cost under deadline constraint by considering the dynamic nature of the cloud computing such as performance and amazon spot instances pricing. In [57], the author propose the SCPOR scheduling algorithm to dynamically schedule a workflow in heterogeneous cloud environment.

The SCPOR algorithm prepares the workflow schedule with the goal of minimizing the makespan by dynamically provisioning and deprovisioning resources of several types with no constraints. In [58], the author propose an adaptive, resource provisioning and scheduling algorithm called WRPS to generate the workflow schedule in a heterogeneous cloud environment. The algorithm has a single objective to minimize the execution cost of the workflow under a user provided deadline constraint by modeling the problem as an unbounded knapsack minimization problem and is based on dynamic programming. The major limitation of this approach is that each partition is considered as an independent bag of tasks and hence only local optimization is performed in each partition of the workflow.

Furthermore, the WRPS algorithm fails to consider the dependencies that exist between the partitions in a workflow, which is a salient feature of the data centric workflows [63, 64]. Another limitation of the WRPS algorithm is that the approach only works for a homogeneous set of tasks in a bag of tasks. However, in reality the data centric workflows consists of heterogeneous tasks and the scheduling optimization is required to consider it [59]. Our BARENTS scheduler is different from the WRPS scheduling algorithm, because we consider the dependencies that exist between the partitions in a workflow through a system generated threshold, credit and debit values. Besides creating initial budget allocation, our approach also dynamically creates the sub-budget adjustment through the runtime estimation. Other dynamic scheduling algorithms [60–62] have also been

proposed. However, these algorithms have only been validated in a simulated environment and not in a real big data workflow system.

CHAPTER 3: THE SHIMMING PROBLEM AND PRIMITIVE WORKFLOW MODEL

3.1 Introduction

In recent years, the term "big data" has become a buzzword in both industry and academia. It is frequently used to refer to the method of processing and analyzing large amounts of heterogeneous types of data to mine hidden patterns, correlations, trends, and other useful information to guide enterprise decisions and strategies for cutting cost or increasing revenues [14, 15]. Scientific workflows, which were originally developed for automating the data analysis process of scientific data (often large in volume and vary in types) to enable and accelerate scientific discovery, are now increasingly used for the processing and analysis of business data as well. However, the challenges of big data, including volume, velocity, and variety pose additional challenges to scientific workflow research.

Scientific workflows consist of one or more computational components connected to each other and possibly to some input data products. Each of these components can be viewed as a black box with well defined input and output ports. Each component is also a workflow, either primitive or composite. Primitive workflows are bound to executable components, such as Web services, scripts, or high performance computing (HPC) services and are viewed as atomic blocks. Composite workflows consist of multiple building blocks connected via data channels. Each of the building blocks can be either a workflow or a data product.

The shimming problem has received much attention in the scientific workflow community [6, 11–13], in which special kinds of adaptors, called shims, are inserted between workflow tasks to resolve the data type incompatibility issue. However, to meet the challenges of big data, several issues in the existing approaches need to be examined. The first issue is to scale the registration and configuration procedure to a large number of workflow tasks, which is critical for big data workflows. The second issue is to ease the integration of a large number of remote Web services and other heterogeneous task components, such as command line executables, that can consume and produce data in various formats and data models into a uniform and interoperable workflow. In particular, we examine automatic shim generation techniques for third-party Web services, whose

development were not workflow-aware. Existing approaches fall short in usability and scalability in addressing these issues.

In this chapter we 1) propose a new simplified single-component based task model based on extensive experiences and lessons learned from our original multiple-component based task model. The new model separates registration from configuration and eases the process of registering external functional components (such as Web services) into p-workflows; 2) propose a shim generation algorithm that elegantly solves the shimming problem raised by Web service based scientific workflows; and 3) we integrate MongoDB , a NoSQL document-oriented database system for storing and managing large-scale unstructured documents. A new version of the DATAVIEW system has been developed to support the proposed techniques and a case study has been conducted to show the feasibility and usability of our proposed techniques.

3.2 Primitive Workflow Model

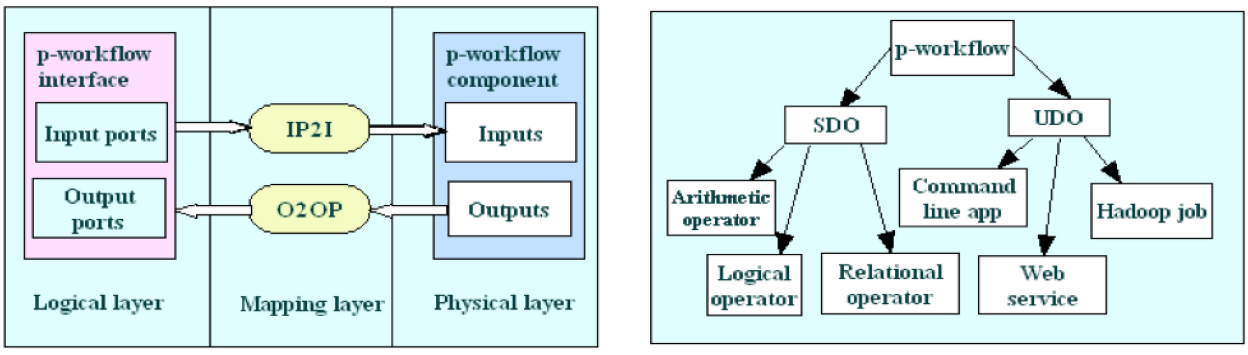
Primitive workflows (a.k.a tasks or p-workflows) are the basic building blocks of a scientific workflow. Composite workflows were used to represent workflows that consist of multiple workflows. Our previously proposed task model [12], contains multiple components inside a single task and hence complicates the registration process as complex shimming needs to be done between the components at design time in order to register any task as a p-workflow. In our new model, we elegantly solve this problem by having only one component inside a task and ease the registration process. At design time, we also provide a configuration mechanism through which we get the input and output port information from the user, when necessary.

One appealing feature of our primitive workflow model is that it provides an abstraction technique, in which different heterogeneous task components can be abstracted into uniform p-workflows so that they can interoperate with one another. While a task manager needs to be aware of the implementation details of each p-workflow so that she can know how to register it as a p-workflow; once registered, such implementation details are hidden from the workflow engineer, who can drag and drop any p-workflow into the design panel, regardless of how it is implemented and connect them with one another into a composite workflow. In our system, we provide a user friendly GUI for design and modification of p-workflows. As shown in Figure 3.1(b), using our model, we classify p-workflows into the following:

1. System Defined operations: They are also called built-in p-workflows. These p-workflows come with a DATAVIEW installation. Examples of such p-workflows include the logical operations (such as AND, OR, and NOT) and relational algebra operators (such as SELECTION, PROJECTION, and UNION).

2. User Defined Operations: These are the p-workflows that are provided by task engineers through registration and configuration. The task engineer provides the task interface and its implementation detail during the registration process. Examples of such p-workflows include command line based application, web service based applications and hadoop based applications.

In order to simplify the workflow design process and to emphasize the usability of our system, we maintain a workflow repository through which our users can easily access existing workflows and use them to compose more complex workflows in the workflow design panel Furthermore, in our model, we separate workflows into two categories: reusable workflows and executable workflows. Reusable workflows are designed without connecting any input and output data products to the workflow, thereby they are mainly used by the workflow engineer to build reusable templates. On the other hand, executable workflows contain input and output data products connected to the input ports of workflows, and they are mainly used by data scientists to perform data analysis and processing. Workflow design is supported through our workflow specification language known as SWL that defines a scientific workflow according to the DATAVIEW Workflow Model [7], through



(a) (b)

Figure 3.1: (a) our proposed primitive workflow model; and (b) a classification of p-workflows.

which one can create scientific workflows with different layers of granularity in a nested manner. SWL, which is automatically created on the server side while designing any workflow, is formulated by translating the workflow diagram that is represented in our workflow visualization language (mxGraph). mxGraph is mainly used at the client side to render and visualize a workflow as a diagram. While a workflow is saved into our system, both the mxGraph and the SWL are serialized and stored into our database. As shown in Figure 3.1(a), our proposed new primitive workflow model consists of the following three layers:

1. Logical Layer: The Logical layer contains the p-workflow interface that models the port details of the workflow such as input port ID, input port type, output port ID, and output port type. At run time, the data flow occurs through the data channel connecting the source workflow constituent to destination workflow constituent.

2. Mapping Layer: The Mapping layer contains a list of mapping information such as mapping of input port of the p-workflow interface to input of the p-workflow component (IP2P) and mapping of output of the p-workflow component to output port of the p-workflow interface (O2OP).

3. Physical Layer: The Physical layer contains only one p-workflow component at a time, that model the services and or application that are used to implement the task. Hence registration of p-workflow is made simple. Heterogeneous characteristics of the p-workflow are captured and modeled in this layer including the inputs, outputs, location of workflow component (such as Web service WSDL file, command line executable file, hadoop jar file).

Using our primitive workflow model, task engineers can easily register a command line based application as a p-workflow by uploading their executable into our DATAVIEW server and providing configuration details, such as input type, input mode, output type, and output mode. As shown in Figures 3.1 and 3.2, we have formulated a mapping table based on different cases that a command line application can fall under in respect to the input and output types. We have identified the following 6 input modes: 1) ByValue - IP2I mapper pass the value from the input port as an input argument during component invocation; 2) ByFile - IP2I mapper generate unique file name, create the file and write content from the input port into the file, pass file name as an argument during

component invocation; 3) ByFixedFile - IP2I mapper create a file with the fixed file name and write content from the input port into the file, pass file name as an argument during component invocation; 4) ByStdin - IP2I mapper get the value from the input port and pipe it to the input stream during component invocation; 5) ByEnv - IP2I mapper will create environment variable with a user driven name and take the value from the input port and write into it during the component invocation; 6) ByConst - IP2I mapper simply add a constant value as one of the argument during component invocation. On the other hand, we have identified the following 3 output modes: 1) ByValue - O2OP mapper simply takes the output that is returned during the component execution which could be either stdout or exit code and bind it to the output port; 2) ByEnv - O2OP mapper will fetch the value from the environment variable using the name provided by user during registration and bind the value to the output port during component execution; 3) ByFile - O2OP mapper will simply fetch the value from the file name which is provided through one of the arguments and bind the value to the output port during component execution.

Input Port name	Input Port type	Cmd Line input	Input Mode
ip1	Integer	arg1	ByValue
ip2	Integer	arg2	ByFile
ip3	Integer	null	ByFixedFile
ip4	Integer	stdin	ByStdin
ip5	Integer	environment variable	Byenv
	null	arg3	ByConst

Table 3.1: IP2I mapping table.

In order to demonstrate the strength of our model, we have implemented and tested the following six cases in which a command line executable can be registered and configured into a p-workflow in our DATAVIEW system:

Case 1: suppose we have a command line java application addition1.class which takes two command line arguments, both of Integer type, and produces the sum of the two integers as the stdout. After converting addition1.class into a p-workflow pw_addition1, the IP2I mapper will take

Cmd Line output	Output Mode	Output Port name	Output Port type
stdout	ByValue	op1	Integer
null	ByFixedFile	op2	Integer
exit code	ByValue	op3	Integer
arg4	ByFile	op4	Integer
environment variable	ByEnv	op5	Integer
arg5	ByFile	null	null

Table 3.2: O2OP mapping table.

the two integers from input ports i1, i2 and then use them as two command line arguments (inputs) for pw_addition1, after the execution of pw_addition1, the O2OP mapper will route the output in the stdout to output port o1 for pw_addition1. Figure 3.2 shows a simple example of a scientific workflow of case1 implemented in our system.

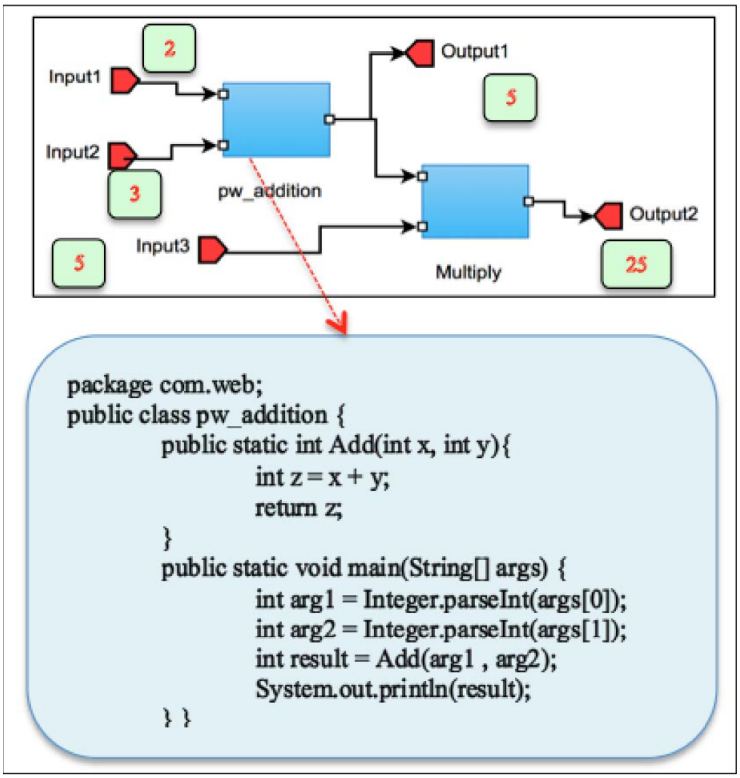


Figure 3.2: Command line based p-workflow.

Case 2: suppose we have a command line java application addition2.class which takes three command line arguments: arg1, arg2, and arg3 where arg1 and arg2 being the input file names that contain an integer, respectively, and arg3 being the output filename that will contain the output sum. After converting addition2.class into a p-workflow pwaddition2, the IP2I mapper will generate three distinct unique file names (strings), arg1, arg2, and arg3. and will take the two integers from input ports i1 and i2 and then create two files with names arg1 and arg2 and write the two numbers into these two files, respectively, after the execution of addition2.class with arguments arg1, arge2 and arg3, a file with name arg3 will be created which contains the output sum, then the O2OP mapper will read the number in file arg3 and route the output to output port o1 for pw_addition2.

Case 3: suppose we have a command line java application addition3.class which takes two command line arguments: arg1, arg2, where arg1 is the first integer and arg2 is the input file name that contains the second integer, and produces the sum of the two integers as the stdout. After converting addition3.class into a p-workflow pw_addition3, the IP2I mapper will take the integer from input port i1 and pass to addition3.class as the first argument arg1, and creates a file called arg2 and writes the second integer from input port i2 to file arg2. After the execution of pw_addition3, the O2OP mapper will route the output in stdout to the output port o1 for pw_addition3.

Case 4: suppose we have a command line java application addition4.class which takes one command line argument: arg1 where arg1 is the first integer and get the environment variable with name env1 that contains the second integer and produces the sum of the two integers as the stdout. After converting addition4.class into a p-workflow pw_addition4, the IP2I mapper will take the integer from input port i1 and pass to addition4.class as the first argument arg1, and creates an environment variable with name env1(that is user driven) and will take the integer from input port i2 and write it into the environment variable. After the execution of pw-addition4, the O2OP mapper will route the output in stdout to the output port o1 for pw_addition4.

Case 5: suppose we have a command line java application addition5.class which takes two command line arguments, both of integer type, and produces the sum of the two integers as the stdout. After converting addition5.class into a p-workflow pw_addition5, the IP2I mapper will take the first integer from input port i1 and use it as the first command line argument (input) and use a fixed value (constant) as the second command line argument (input) for pw_addition5, after the

execution of pw_addition5, the O2OP mapper will route the output in the stdout to output port o1 for pw_addition5.

Case 6: suppose we have a command line java application addition6.class which takes two stdin (standard command line inputs) and convert it into integer type, and produces the sum of the two integers as the stdout. After converting addition6.class into a p-workflow pw_addition6, the IP2I mapper will take the first integer from input port i1 and pipe it into the input stream as the first stdin (input) and take the second integer from input port i2 and pipes it into the input stream as the second stdin (input) for pw_addition6, after the execution of pw_addition6, the O2OP mapper will route the output in the stdout to output port o1 for pw_addition6.

We identify the following advantages of our primitive workflow model:

1. Domain Proficiency: Since the developers of these third party applications are experts in the domain and the software components are well studied and implemented, using it within a scientific workflow is a big advantage and simplifies the workflow design process through the anatomy of reusability and sharing.

2. Reliability: These third party applications are highly reliable because they are used by many scientists all over the world and hence most of the problems/bugs in the software will be fixed promptly.

3. Infrastructure: Most of third party applications such as Web services are hosted in a remote server and hence high computation cost is not an issue as the infrastructure is available free through invoking the Web service.

4. Support: Many third party applications are free to download, install and also have a forum where people discuss various issues and fixes about these software components.

3.3 Shim Generation Algorithm

In this section, we first propose an approach to solve the shimming problem in our DATAVIEW system, that occurs during registration of any Web service based p-workflow through a user-driven configuration mechanism and then provide a shim generation algorithm to translate a p-workflow into a composite workflow by wrapping two special adaptors known as pCombiner and pSplitter

shims around the p-workflow. Next, we demonstrate an example of how our GenShims algorithm works. Because of the heterogeneous nature of scientific workflows, our system provides a platform to register primitive-workflows from different sources such as Web services, command line applications. However, this then becomes an issue because of the incompatibility of data types between the input/output port of a p-workflow and input/output of component connected to it. Due to the shimming problem that exists during the workflow composition, at run time the p-workflow is not interoperable with other types of workflow constituents. Through a user driven configuration mechanism, we separate the shimming problem from registration and thereby making p-workflow registration simple and usable. After registering the p-workflow, the shimming problem is solved by configuring the input and output port of any arbitrary p-workflow and the system automatically creates instances of two special kinds of shims known as pCombiner and pSplitter. Finally a new wrapper workflow is created on the fly during the p-workflow configuration with the instances of pCombiner and pSplitter inserted into the new wrapper workflow. In this way we hide the shim at the workflow level and abstract the existence of shim from the data scientist perspective, but during workflow execution, a shim automatically does the conversion based on the configuration details provided by the workflow designer. Data scientists would simply need to drag and drop the wrapper workflow, connect data products and execute the workflow to view the results.

In our DATAVIEW system, a task engineer can easily register any arbitrary Web service operation by providing the system with the WSDL file of any arbitrary Web service. We have implemented WSDL parser functionality within our system that can automatically get the input and output of an operation from the WSDL and convert it into the input and output port of the workflow. Additionally we have generated automatically two special shims: pCombiner and pSplitter. pCombiner shim, is a special kind of shim adaptor that takes in any number of arbitrary data from the input ports of arbitrary types and create a new data product of type XML and bind the XML data to the output port. The pSplitter shim, is another special kind of shim adaptor that takes in data from the input port of type XML and extract the elements inside it in a unique manner and split the result into multiple outputs, which are then bound to multiple output ports. Input port details of the pCombiner such as (no of input ports, input port types) and output port details of the pSplitter such as (no of output ports, output port types) are identified during the p-workflow

configuration. As shown in Figure 3.5, the proposed GenShims algorithm takes an input p-workflow ws1 and converts it into a composite workflow c-ws1. In order to create the composite workflow c-ws1, our algorithm creates both workflow specification (SWL) and visualization (mxGraph) for the workflow automatically. Our pCombiner is IP2I mapper for Web service based p-workflow and each time GenShims algorithm is executed, it creates a new instance of pCombiner. On the other hand, the pSplitter shim is an O2OP mapper for Web service based p-workflow and each time the GenShims algorithm is executed, it creates a new instance of pSplitter. Figure 3.4 shows the SWL of the workflow with shim inclusion in our DATAVIEW system.

Our approach currently works for Web services with primitive types of int and string for the following cases of web service operation: one input and one output, one input and multiple outputs, multiple inputs and one output, multiple inputs and multiple outputs, no input and one output, no input and multiple outputs, no input and no output, one input and no output, multiple inputs and no output. In the future, we will explore the above cases with complex data types as inputs and outputs.

In Figure 3.3, we show an example of shim generation for Web service based workflows. To simply the explanation, we have developed a Web service ws_addsubtract in Java, which takes a pair

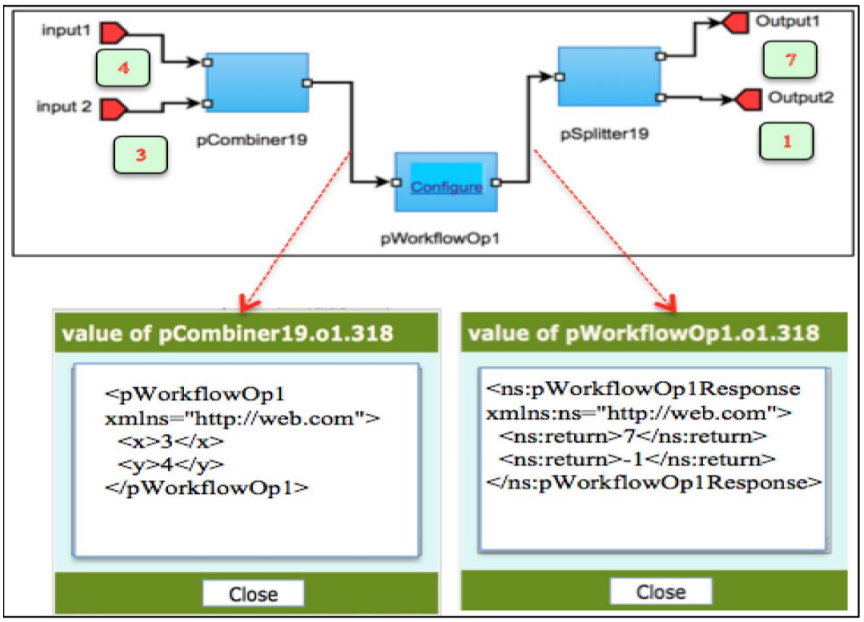


Figure 3.3: Web service based p-workflow.

```
<workflowSpec>
<workflow name="ws_pWorkflowOp1" root="true">
<workflowInterface>
 <inputPorts>
<inputPort>
<portID>i0</portID>
<portName>i0</portName>
<portType>Integer</portType>
</inputPort>
<inputPort>
<portID>i1</portID>
<portName>i1</portName>
<portType>Integer</portType>
</inputPort>
</inputPorts>
 <outputPorts>
<outputPort>
<portID>o0</portID>
<portName>o0</portName>
<portType>Integer</portType>
</outputPort>
<outputPort>
<portID>o1</portID>
<portName>o1</portName>
<portType>Integer</portType>
</outputPort>
</outputPorts>
</workflowInterface>
<workflowBody mode="graph-based">
<workflowGraph>
<workflowInstances>
<workflowInstance id="pWorkflowOp1">
<workflow>pWorkflowOp1</workflow>
</workflowInstance>
<workflowInstance id="pCombiner19">
<workflow>pCombiner19</workflow>
</workflowInstance>
<workflowInstance id="pSplitter19">
<workflow>pSplitter19</workflow>
</workflowInstance>
</workflowInstances>
<dataChannels>
<dataChannel from="pWorkflowOp1.o1" to="pSplitter19.i1"/>
<dataChannel from="pCombiner19.o1" to="pWorkflowOp1.i1"/>
</dataChannels>
</workflowGraph>
<G2W>
<inputMapping from="this.i0" to="pCombiner19.i1"/>
<inputMapping from="this.i1" to="pCombiner19.i2"/>
<outputMapping from="pSplitter19.o1" to="this.o0"/>
<outputMapping from="pSplitter19.o2" to="this.o1"/>
</G2W>
</workflowBody>
</workflow>
</workflowSpec>
```

Figure 3.4: Workflow Specification (SWL) with shim inclusion.

Algorithm: GenShims
Input: Primitive Workflow ws1
Output: Composite Workflow c_ws1
Begin

1. **If** ws1 is of type web service
2. **Then** extract configuration details such as input port name{In), input port type(It), output port name(On), output port type(Ot) by parsing the wsdl file of the primitive workflow ws1.
3. Auto populate port details in the configuration section of the primitive workflow ws1, with an option for the users to modify or add new port information.
4. Create a new instance of the combiner shim (C) , which takes in arbitrary number of input ports of name In and type It and combine them into an output Oc of type XML.
5. Create a new instance of the splitter shim (S) , which takes in XML input Is and parse the XML to extract all the values inside it and split them into a series of outputs of name On and type Ot.
6. Create SWL for composite workflow c-ws1 with the workflow interface that has input ports In with port type It and output ports On with port type Ot.
7. Append the workflow instances section to SWL that was created in step 6 with the instances pCombiner, ws1,and pSplitter.
8. Append the dataChannel section to SWL with the following source -> destination data flow construct.
    a) Output from the combiner shim (Oc) to the input of the primitive workflow ws1.
    b) Output of the primitive workflow ws1 to the input of the splitter shim (Is).
9. Create the mxGraph diagram for the composite workflow c_ws1 with the input port In of type It connected to pCombiner input port. Output of pCombiner connected to the input of ws1. Output of ws1 connected to the input of pSplitter. Output of pSplitter connected to the output port On of type Ot
10. **Else**
11. No shim required to be inserted.

End Algorithm

Figure 3.5: GenShims Algorithm.

of integers a and b, and returns their sum and difference, respectively as two integers c and d. After registering the WSDL URL of this Web service. (http://dmsg2.cs.wayne.edu/axis2/services/pWorkflow1?wsdl) in DATAVIEW, the system automatically extracts all the operations of the Web service. In our case, the user selects the only operation available, pWorkflowOp1, and the system will automatically convert it into p-workflow pWorkflowOp1. This workflow has one single input port of XML type and one single output port of XML type. The user can then click the configure link in this workflow, which allows the user to choose the number of input ports (which is 2) and the number of output ports (which is also 2). After the completion of the configuration, two shims, pCombiner19 and pSplitter19, will be generated automatically, and a new complex workflow cw_pWorkflowOp1 is constructed. Workflow cw_pWorkflowOp1 will have two input ports a and b, and two output ports c and d.

3.4 Integration of NoSQL Database

NoSQL databases are distributed and schemaless databases that have recently emerged as a technology developed to address the need to store and process huge volumes of data. Existing NoSQL databases such as Riak, MongoDB , Cassandra, HBase, Neo4j enable horizontal scalability across a large number of commodity servers. In DATAVIEW, we have integrated MongoDB within our data product manager to store and manage different data products. MongoDB is a scalable, open source document-oriented database that is commonly classified as a NoSQL database. MongoDB uses JSON-like documents with no schema to store data inside different collections. In DATAVIEW, execution of a scientific workflow consumes and produces huge amount of distributed data objects. These data objects are heterogeneous of various data types. A scientific workflow might use mixed data types as the data could be any form and aligns well with the schemaless and scalable nature of MongoDB . In DATAVIEW, we provide an abstraction of data objects stored in MongoDB as a data product.

Three challenges of Big Data namely the volume, velocity and variety is a key issue focused today in both academia and industry. Data can be from different sources and the system should have the capability to some how register and import the data and perform analysis and process the data. One of the main advantage of integrating MongoDB in our system is that, it is horizontabally scalable across the commodity servers and hence the data from the scientists can be shipped through our

DATAVIEW Data Type	Description
Scalar	to store data of following types: string, decimal, integer, non-positive integer, non-negative integer, positive integer, negative integer, unsigned int, unsigned long, unsigned short, unsigned byte, double, float, long, int, short, byte, boolean.
Relational	to store relation that are represented as tables.
Collectional	to store collection that are represented as key-value pairs.
XML	to store xml documents that conforms to well formed xml value
File	to upload the file on server side and store physical path of the file.

Table 3.3: Data Types supported in DATAVIEW.

data product manager with high performance. Another advantage is MongoDB offers MapReduce support at the database level to process huge amount of data into meaningful aggregated results. MongoDB applies the map phase to each document process the data.

One of the main advantage of integrating MongoDB in our system is that, it is horizontabally scalable across the commodity servers and hence the data from the scientists can be shipped through our data product manager with high performance. Another advantage is MongoDB offers MapReduce support at the database level to process huge amount of data into meaningful aggregated results. MongoDB applies the map phase to each document inside the collection that match the query condition and emits a (key, value) pairs. And then the reduce phase is run for those keys that contain multiple values and data is collected and aggregated to form a new collection in the database. Our solution to address the challenge occurred by such a large variety of types in big data is to classify the different data types into a custom defined approach and then perform shimming to transform data into one of the DATAVIEW data types. As shown in Table 3.3, we defined the notion of DATAVIEW data type in our system and broadly classify data products into the following categories: 1) scalar 2) relational 3) collectional 4) XML and 5) file.

3.5 Case Study

In our DATAVIEW system, we have created several automotive consumer review big data primitive workflows in order to collect, merge, extract, and analyze the raw data from automotive review websites, including KBB, Edmunds, and MSN Autos. As part of our case study, we deployed

a MongoDB sharded cluster in FutureGrid that provides a schemaless, horizontabally scalable, and MapReduce enabled data storage and processing in a cluster enivronment. In order to deploy the MongoDB cluster, we created one VM instance with 4 GB memory and 40 GB disk space and have setup the master node where the mongos is running. In addition, we created 3 VM instances with each 2 GB memory and 40 GB disk space and have setup the config servers which contains the metadata information such as shard and block level information. In MongoDB, the config servers are single point of failure and hence we created 3 instances for replication purpose on node failure. Finally we created 3 more VM instances with each 4 GB memory and 60 GB disk space and have setup the mongod shards that contains the actual data sharded (split inside chunk and new chunks). In addition to deploying the sharded cluster, we have also integrated the MongoDB Java driver within our DATAVIEW system, in order to make a connection with the MongoDB cluster and perform several operations on the data.

As shown in Figure 3.6, we created an automotive consumer review analysis workflow. Firstly, we created a non map reduce p-workflow known as SearchNonMR that takes in three inputs (AutoReview, FieldName1, SearchTerm1) and query the FieldName on the master node of the MogoDB cluster with an or condition to find the match for the list of search terms provided by the user such as "MPG, Fuel Economy, Excessive comsumption". During our experiments we found that the execution of the workflow is time consuming and takes around 629 seconds for inserting 8425 (11.25 MB) records into a new collection in the MongoDB cluster. Secondly, we created two map reduce p-workflows known as SearchLevel1MR and SearchLevel2MR to process the consumer review dataset in a map reduce manner and break down the data into State and City level. SearchLevel1MR takes in 3 inputs (AutoReview, FieldName2, SearchTerm2) and execute the map reduce program written with a mapper that emits (Search Term, Review) based on a REGEX match condition and a reducer that aggregates all the review for distinct search terms. Output of the map reduce function is automatically binded to a new collection inside the MongoDB cluster. Output from the SearchLevel1MR is passed as input to SearchLevel2MR which would execute a map reduce program with a mapper that loops through each item in the aggregated review and for each review item it emits for example (State Name, Review) based on a REGEX match condition and a reducer aggregates all the reviews for distinct states. During our experiments we found that the execution

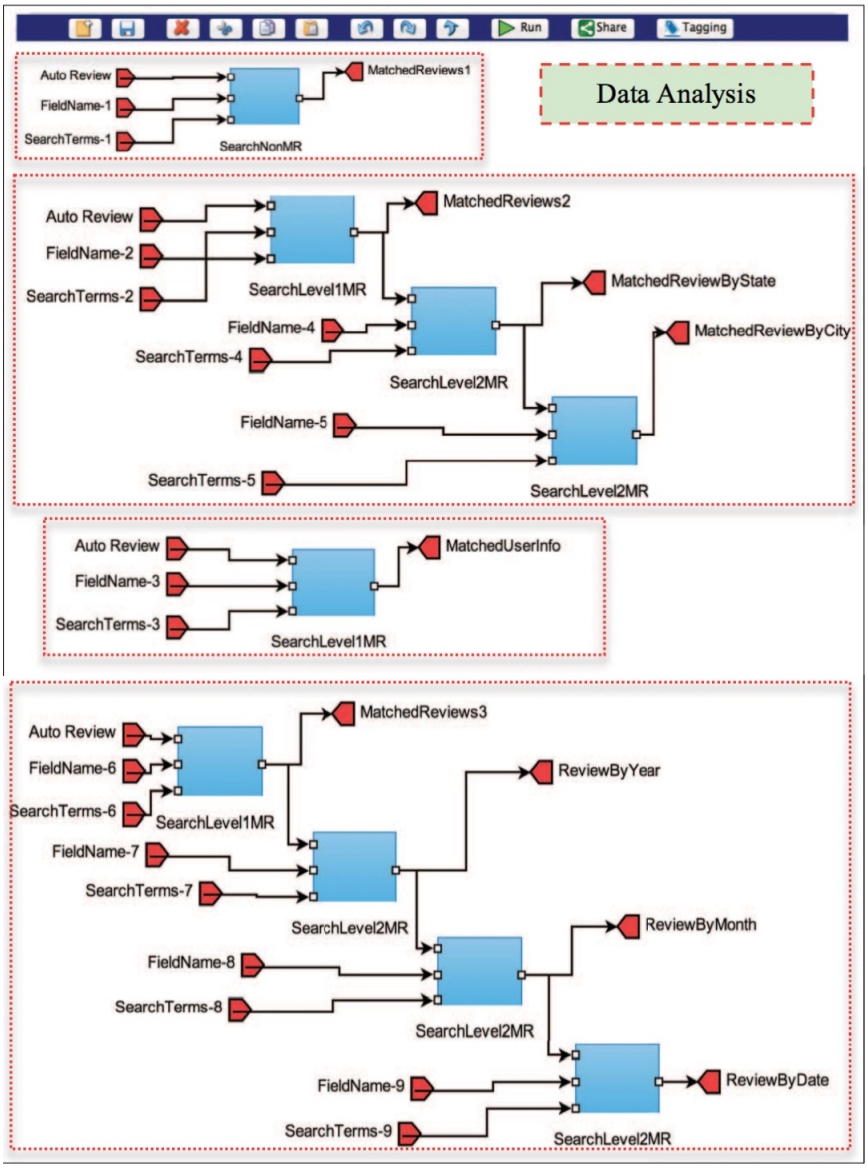


Figure 3.6: An Automotive Consumer Review Analysis Big Data Workflow.

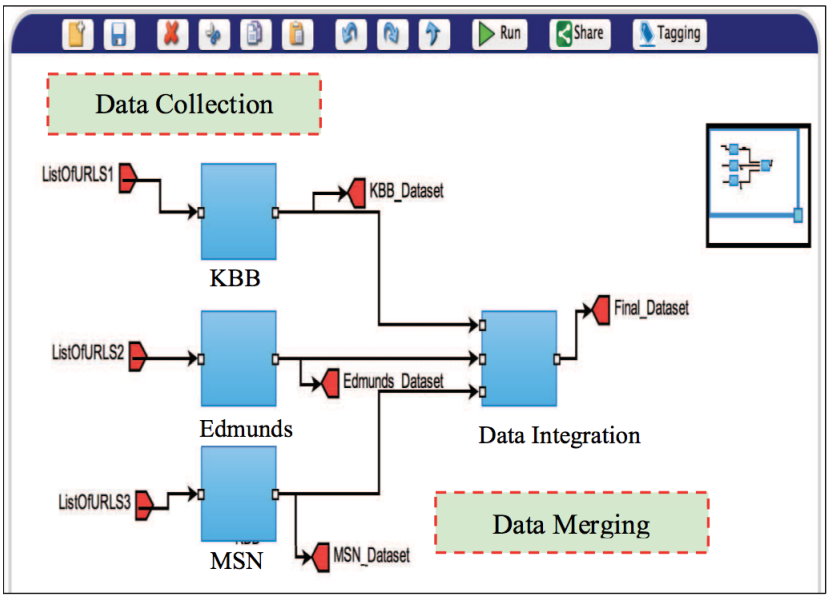


Figure 3.7: An Automotive Consumer Review Data Collection/Integration Workflow.

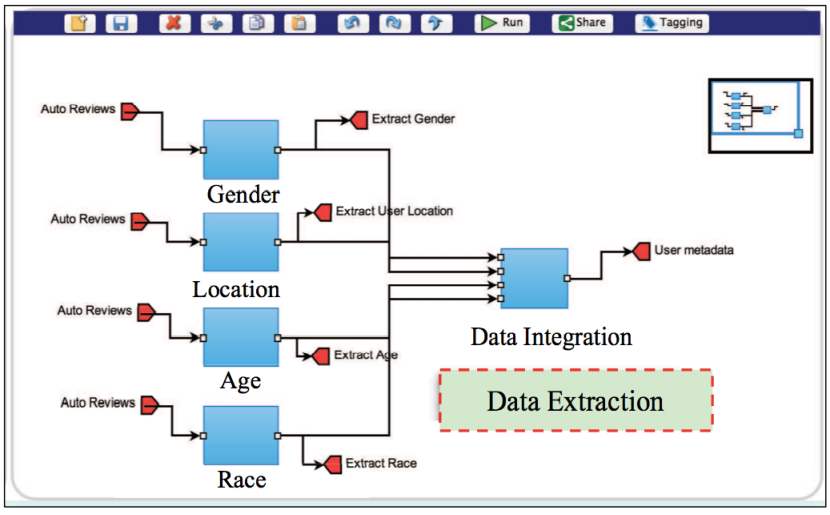


Figure 3.8: An Automotive Consumer Review User Meta Data Extraction Workflow.

of the workflow takes around 17 seconds for creating a output collection of size 6 MB. Thirdly, we created a workflow to get the list of users who posted reviews in a list of states. During workflow execution a map reduce program is executed to get the list of reviews that contain the match for one or more of the items in the list of states and the aggregated review are stored into a new collection in MongoDB. Fourthly, we created a workflow to break down the review into Year, Month and Date and hence the data scientist can easily visualize how many reviews were posted for a particular year, month, date etc.

In order to perform data collection, as shown in Figure 3.7 we created a series of command line based p-workflows known as KBB, Edmunds and MSN where data crawling procedures are executed internally and passed into a customized HTML parser, to extract consumer review information. Output from the three data collection workflows are passed as input to the data integration workflow where the data is integrated together as one collection by executing the union operation on the input dataset. As shown in Figure 3.8, we created data extraction p-workflows such as Gender, Location, Age and Race to extract the users me-ta data information automatically from the consumer reviews. Due to space limitation, we have not included the details of the implementation for data extraction p-workflows.

3.6 Chapter Summary

In this chapter, we have proposed a new simplified single-component based task model based on extensive experiences and lessons learned from our original multiple-component based task model. Further, we have proposed our shim generation algorithm that elegantly solves the shimming problem raised by Web service based scientific workflows. Furthermore, we discussed the experiences from integrating MongoDB , a NoSQL document-oriented database system for storing and managing large-scale unstructured documents.

CHAPTER 4: A FOLKSONOMY BASED SOCIAL RECOMMENDATION SYSTEM FOR WORKFLOW REUSE

4.1 Introduction

In the past decade, scientific workflow systems have significantly improved scientists ability to structure scientific processes, use computational resources, and analyze their data more efficiently. Such productivity can be further enhanced by sharing, reusing, and repurposing existing tasks and workflows across different users and institutes. However, existing scientific workflow systems are mainly single-user oriented with limited sharing and reusing functionalities. To overcome such limitations, we propose a folksonomy-based social workflow recommendation system to improve workflow design productivity.

The past decade has witnessed the growing benefits of using scientific workflow systems to improve the productivity of designing scientific processes and data-driven scientific discoveries in various domains, such as bioinformatics [22], neuroinformatics [23], ecology [24], oceanography [25], astronomy [26], and high-energy physics [27]. However, the productivity of workflow design is still hampered in existing scientific workflow systems in two ways. Most existing scientific workflow management systems are single-user oriented and thus across user sharing and reuse cannot be achieved within the system per se. Meanwhile, workflow design is largely still a tedious and error-prone process with little or no automation support. While social scientific workflow sharing environments, such as MyExperiment [28], greatly facilitate workflow sharing and reuse across tools, users, and institutes, such sharing is external to a workflow design environment and thus provides little help to the design of an in-progress workflow.

In the meanwhile, folksonomy, the practice and method of collaboratively creating and reusing tags to annotate and categorize digital contents, has become a key characteristic of Web 2.0 [18]. In contrast to a taxonomy, which has a fixed vocabulary, a folksonomy allows each author or user to create his or her own terms contributing to an evolving folksonomy. Such flexibility greatly improves the productivity of tagging and annotation and engagement of users. As a result, more digital contents are annotated and searchability is improved. Moreover, a folksonomy keeps track of emerging trends in tag usage and user interests. Therefore, it is natural to adopt folksonomies to annotate and categorize scientific workflows.

To overcome the above limitations of existing workflow design and sharing systems, in this chapter we propose a folksonomy-based social workflow recommendation system to improve workflow design productivity. In this chapter, we i) developed a web-based workflow design environment (called Web-bench) to allow users to create workflows and collaboratively annotate and categorize them using social tags. The resulted folksonomy improves workflow searchability and shareability. ii) proposed several workflow recommendation strategies to automatically or semi-automatically augment an in-progress workflow, leveraging both structural and semantic similarities between workflows and guiding information extracted from previously created workflows in the database. iii) implemented the proposed environment and strategies in a prototype based on the DATAVIEW scientific workflow management system and validated our approach with numerous use cases.

4.2 An Overview of Workflow Recommendation Framework

Our proposed workflow recommendation framework is used to improve workflow design productivity by recommending a suitable workflow that is both syntactically and semantically compatible to any incomplete in-progress workflow. In accordance with the reference architecture for scientific workflows [7, 63], we propose a workflow design inspector, a syntactic recommender and a semantic recommender as the core components of the workflow recommendation framework, which are positioned in the Webbench and the Workflow Engine, two subsystems in the reference architecture. In Figure 4.1(a), we show the system architecture for DATAVIEW [63], which is composed of seven loosely coupled subsystems, including the Webbench, the workflow engine, the workflow monitor, the cloud resource manager, the data product manager, the provenance manager, and the task manager. In Figure 4.1(b), we show an overview of the workflow recommendation framework, in which two recommenders, the syntactic recommender and the semantic recommender, are located in the workflow engine and the workflow design inspector in the Webbench, respectively.

The Webbench in Figure 4.1(a) features an online scientific workflow system that allows data scientists to create, edit and run a visual scientific workflow online. In our DATAVIEW system, we use mxGraph, a visualization language program, for representing the workflow design. During workflow design, every time a new workflow is added to the workflow design panel, the workflow design inspector extracts the list of complete and incomplete workflows that exist in the workflow design panel. The workflow design inspector is the key compo-nent that drives the

workflow recommendation framework. During the workflow design process, the workflow design inspector provides a clickable button called "Recommend Workflow" within the incomplete workflow and sends the specification of the incomplete workflow to the workflow engine.

The workflow engine in Figure 4.1(b), on the other hand, invokes the SWL Parser to extract the specification of the incomplete workflow that is provided by the workflow design inspector. Specification of the workflow contains logical details, mapping details, and physical details of the workflow. Logical details include the workflow name, the input, and the output ports of the workflow. Mapping details include the mapping information that illustrates how the data product is mapped to the input and the output port of the workflow. Physical details include information such as the location of the code that is embedded inside the workflow. The syntactic recommender component accepts the specification of the incomplete workflow and validates the connectivity on the ports to check whether incompleteness is on either the input side or the output side of the workflow. Incompleteness on an input requires a producer workflow whose output port shall be connected to the input port of the incomplete workflow. Incompleteness on an output port requires a consumer workflow whose input port shall be connected to the output port of the incomplete workflow. The syntactic recommender then performs a look up in the workflow repository to find the list of workflows that contains an input port that matches the output port of the incomplete workflow. A list of workflows that match with the port information of the incomplete workflow is then added to the

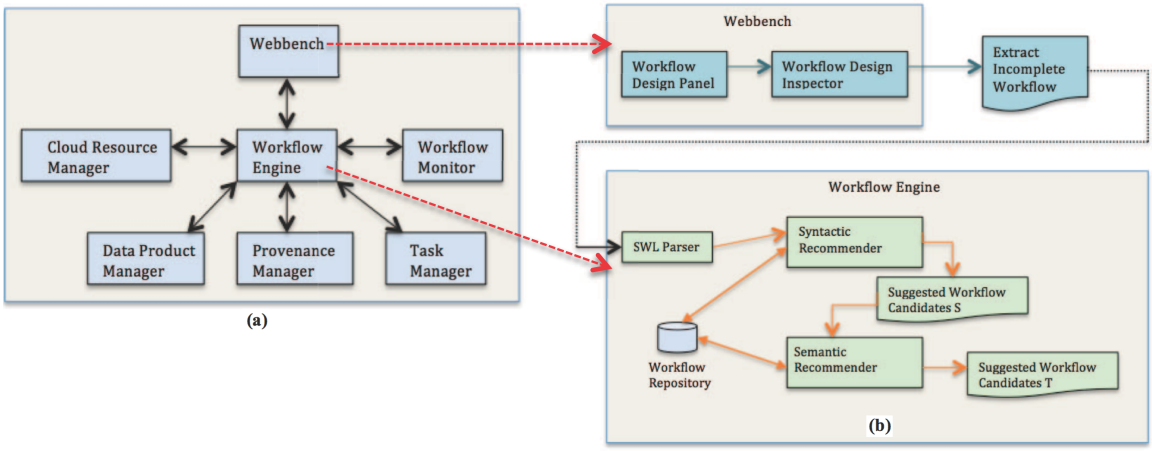


Figure 4.1: (a) DATAVIEW Architecture; (b) Workflow recommendation framework.

suggested workflow candidate list S that contains both the recom-mended producer and consumer workflows.

Although the list of workflows in the suggested workflow candidate S is syntactically compatible with the incomplete workflow, those workflows might not be semantically relevant to the incomplete workflow nor preferred by the user. Hence, the semantic recommender is used to filter and identify a sublist of workflow candidates from S that are also semantically compatible by leveraging the tag annotations. The semantic recommender component com-putes a workflow recommendation score based on both workflow similarity score and user interest matching score between the incomplete workflow and each of the workflows in the suggested workflow candidates S. with that of the tags associated with the incomplete workflow. After computing the workflow recommendation score, we rank the workflows based on the recommendation score and a new list of producer and consumer workflows is added to the suggested workflow candidate list T and finally recommended to the data scientist.

4.3 A Folksonomy Based Workflow Model

Folksonomy [16] is a classification system derived from the practice and method of collaboratively creating and managing tags to annotate and categorize content. As shown in Figure 4.2(c), in folksonomy, a user tags a resource (e.g. workflow) and the ternary relationship between the user, the tag and the resource is collectively known as tag assignment (TAS).

As shown in Figure 4.2(d), TAS can be represented as a graph consisting of a set of users, a bag of tags and a set of workflows, thereby forming a folksonomy relationship. TAS emphasizes both user preference and workflow tag relevance factors. In Figure 4.2(b), we show the user preference, which is used to compute how much a particular user prefers the tag based on the users previous tagging activities. In Figure 4.2(a), we show the workflow relevance, which is used to compute how much a par-ticular tag is relevant to the workflow based on the tagging activities on the workflow.

Our previously proposed workflow model [59] contains the syntactic information of the workflow such as name, input port details, output port details and data mapping details. However, it does not include any semantic information about the workflow except the workflow name. In our new model, we support tagging the workflow during the workflow design process. Tags represent lightweight textual information that provides more insights about the workflow and more impor-

tantly it is user driven. Hence the semantic information driven by the tags is leveraged to provide support to the user by increasing the users workflow design productivity.

Definition 4.1: A Folksonomy is a tuple F = (U, T, W, Y) where U, T, and W are finite sets, whose elements are called users, tags and workflows, respectively. Y is a ternary relation between them, i. e.,

$$Y \subset U \times T \times W$$

whose elements are called tag assignments. For workflow wf_i, we use tags(wf_i) to denote the bag of tags (duplicates are included) annotated by all users for workflow wf_i. For a user u_m, we use profile(u_m) to represent the bag of tags that a user u_m used to annotate all workflows in the workflow repository.

ASSUMPTION 1: Given an incomplete workflow wf_i that is being created by a user u_m, the rank of a workflow wf_j in the recommended list of workflows is decided by two scores, WS(wf_i, wf_j), the similarity matching between wf_i and wf_j, and IM(u_m, wf_j), a user profile interest matching between user u_m and workflow wf_j.

A scientific workflow represents a multiple-step data analysis pipeline that chains several data analysis workflows (e.g. Web Services, Command line applications) together via data links, which connect the output of one workflow to the input of another workflow. We have two types of workflow namely, a primitive workflow, that has no subworkflows inside it, a composite workflow,

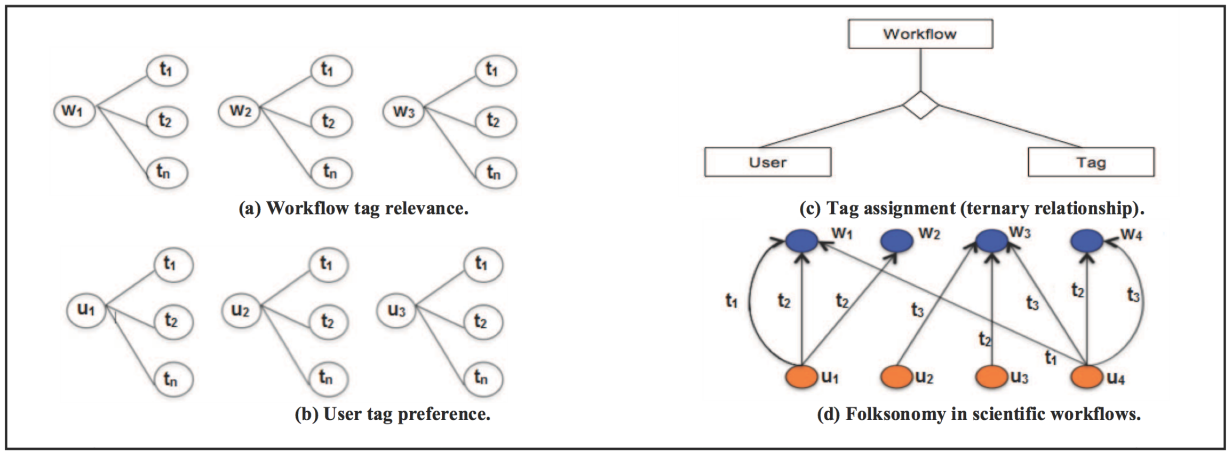


Figure 4.2: An overview of folksonomy in workflow model.

that has at least one or more subworkflows inside it. More formally a scientific workflow is defined as:

Definition 4.2: A scientific workflow W is a tuple (u, T, TS, IP, OP) where u is the unique identifier of the user who created W, T is a bag of tags that are assigned to W, TS is the set of constituent subworkflows inside W, IP is the set of input ports of W, and OP is the set of output ports of W. W is primitive if

$$|TS| = \emptyset$$

and composite otherwise.

The tf-idf score is widely used in the information retrieval community to identify how important a particular word is to a document in the collection. In our workflow model, we refer documents as workflows and terms as tags, and hence, the tf-idf score is used to compute how important a particular tag is in the context of a workflow. The tf-idf score is computed by multiplying term frequency (tf) with inverse document frequency (idf). The term frequency (tf) is used to compute the frequency of the particular tag $t_k$in a workflow $wf_i$and is computed by the equation below:

$$tf(t_k, wf_i) = 0.5 + \frac{0.5 \times f(t_k, wf_i)}{max\{f(t, wf_i) : t \in wf_i.T\}}$$

where f(t_k,wf_i) is the raw frequerncy of tag t_k in wf_i.T. The inverse document frequency (idf) is used to measure how much common a particular tag t_k is in the N number of workflows in the workflow repository W and is computed by the equation below:

$$idf(t_k, W) = log\frac{|W|}{|\{wf_i \in W : t_k \in wf_i.T\}|}$$

$$tf - idf(t_k, wf_i, W) = tf(t_k, wf_i) \times idf(t_k, W)$$

Given a user u_m and all the users U in the system, let u_m.T be the bag of tags used by user u_m to annoate all the workflows in W, then the tf-idf of a tag t_k with respect to user u_m and U, td-idf(t_k, u_m, U), can be defined similarly.

Definition 4.3: Workflow Similarity WS is used to compute the similarity between any two arbitrary workflows. Given two workflows wf_i and wf_j, we represent them as two vectors:

$v(wf_i) = (w_{i1}, w_{i2}, w_{i3}, ..., w_{in})$ $v(wf_j) = (w_{j1}, w_{j2}, w_{j3}, ..., w_{jn})$

where w_{ik} = tf-idf(t_k, wf_i, W), w_{jk} = tf-idf(t_k, wf_i, W) and n is the number of tags in T. The workflow similarity between wf_i and wf_j is computed by the equation below:

$$WS(wf_i, wf_j) = \frac{\sum_{k=1}^{n} w_{ik} \times w_{jk}}{\sqrt{\sum_{k=1}^{n} (w_{ik})^2} \times \sqrt{\sum_{k=1}^{n} (w_{jk})^2}}$$

Based on assumption 1, a recommended workflow should be not only similar to the incomplete workflow, but also matching to the interest of the user u_m, which is characterized by her profile to achive personalized recommendation. To this end, we introduce the notion of user interest matching score.

Definition 4.4: The User Interest Matching Score IM is used to compute the interest matching degree between a user u_m and a workflow wf_j, and is defined as follows:

$$IM(u_m, wf_j) = \frac{\sum_{k=1}^{n} w_{jk} \times u_{mk}}{\sqrt{\sum_{k=1}^{n} (w_{jk})^2} \times \sqrt{\sum_{k=1}^{n} (u_{mk})^2}}$$

where w_{jk}= tf-idf(t_k, wf_j, W), u_{mk} = tf-idf(t_k, u_m, U) and n is the number of tags in T.

Definition 4.5: Workflow Recommendation Score WR is used to rank the workflows that are part of the suggested workflow candidates S provided by the syntactic recommender component by computing the workflow similarity and user profile similarity with respect to the incomplete workflow. (See section 4.2). Based on the workflow recommendation score computed by the equation provided below, the list of workflows are added to the suggested workflow candidates T and recommended to the user.

$$WR(u_m, wf_i, wf_j) = \gamma.WS(wf_i, wf_j) + (1 - \gamma).IM(u_m, wf_j)$$

where, γ is the recommendation weight factor (RWF) that is used to balance workflow similarity

score and user interest matching score that satisfies

$$0 \geq \gamma \leq 1$$

4.4 Workflow Recommendation Algorithms

Our workflow recommendation framework considers both syntactic and semantic information that are part of the workflow in order to recommend a suitable workflow to the incomplete workflow. As part of the workflow design process, we provide the user with the option to annotate the workflow by clicking on the tagging button in the workflow design panel. As shown in Figure 4.2(d), the following tag assignments are created in our user profile table during the user annotation process. (u_1,t_1,w_1), (u_1,t_2,w_1), (u_1,t_2,w_2), (u_2,t_2,w_3), (u_3,t_2,w_3), (u_4,t_3,w_1), (u_4,t_3,w_3), (u_4,t_2,w_4), (u_4,t_3,w_4)

As evident from the above tag assignments, in our system different users can use the same tags to annotate the same workflow. Also different users can use different tags to annotate the same workflow. Because of these constraints, we allow duplicates to be included and hence represent the tags as bag of words.

The syntactic workflow recommender component is mainly used to compare the input/output ports of the incomplete workflow with the workflows residing in our workflow repository. The incompleteness in the workflow occurs mainly due to the missing connection link from the input ports of the workflow to another producer workflow or output ports of the workflow to another consumer workflow. Producer workflows are those workflows in which one or more of the output ports of the workflow are connected to one or more input ports of the incomplete workflow. Consumer workflows are those workflows in which one or more of the input ports of the workflow are connected to one or more output ports of the incomplete workflow. As shown in Algorithm1, Syntactic workflow recommender component accepts two inputs as, incomplete workflow and the list of workflows in the repository.

In our DATAVIEW system, we use an XML based workflow specification language called SWL to represent the meta data information of the workflow. Each workflow in our repository contains a SWL associated with it. We implemented SWL Parser as part of the workflow engine to parse the specification file of the workflow and get all the input and output ports associated with the

workflow. Output of the Algorithm1 generates two sets of recommended workflow lists for producer workflows and consumer workflows. For each logical port in the incomplete workflow, we compare the input ports of the incomplete workflow to find a suitable workflow in the workflow repository that contains at least one matching output port that is type compatible with that of the input port of the incomplete workflow. The workflow is then added to the list of recommended producer workflows. We compare the output ports of the incomplete workflow to find a suitable workflow in the workflow repository that contains at least one matching input port that is type compatible with that of the output port of the incomplete workflow. The identified workflow is then added to the list of recommended consumer workflows.

For example, in Figure 4.3 we show an in-progress workflow that contains an incomplete composite workflow w_3, with the input port connected to another producer workflow w_2. But the output port of the workflow is not connected to any workflow and hence we provide the user with a workflow recommendation link. The output port of the incomplete workflow w_3 is integer type and hence we look at the workflows in the workflow repository and identify those workflows that contain at least one matching port. As shown in Figure 4.3(b) and Figure 4.3(c), the sample recommended workflows wf_1 and wf_2 contains at least one port of type integer. Hence both wf_1 and wf_2 are added to the list of consumer workflows as part of the recommendation list generated by the syntactic workflow recommender.

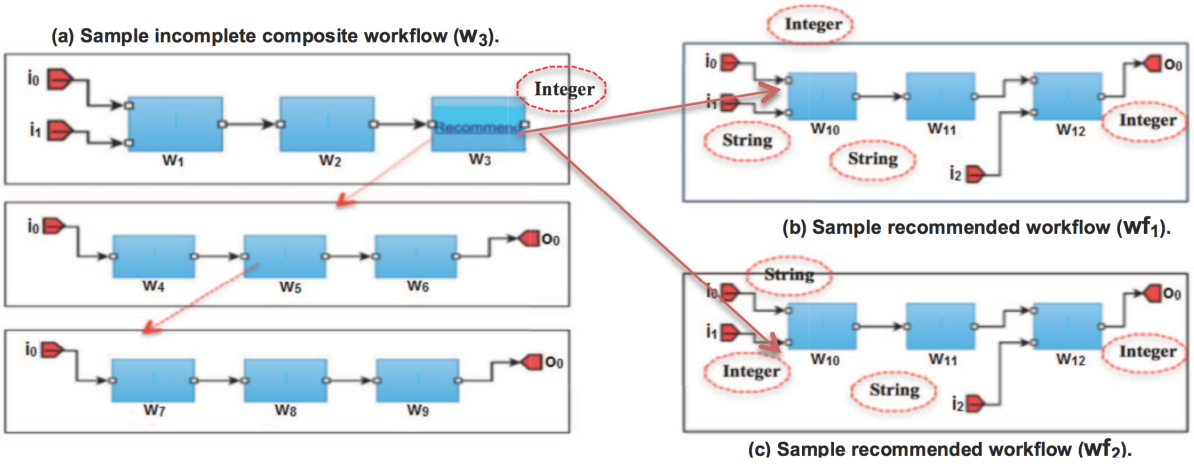


Figure 4.3: An example workflow.

Although the workflows wf_1 and wf_2 are syntactically compatible with w_3, the workflows might not be relevant or preferred by the user based on the users profile. So, we identify the workflow similarity by using Algorithm2 to compute the similarity between the vector representation of the incomplete workflow and the vector representation of the recommended workflows generated as output of the syntactic workflow recommender component. In addition to the workflow similarity, we also compute the user interest matching score and the workflow recommendation score. Based on the recommendation score and a system defined threshold value, the workflow ranking is computed and the list of suggested workflow candidates for both the producer and consumer workflows is generated and provided to the user. We set a system defined threshold value, so that only those workflows that contain the recommendation score greater than the threshold value are added to the recommendation list.

By setting a threshold value, we avoid recommending any workflow that has a low recommendation score to the user. One challenge incurred during our workflow recommendation technique is for folksonomy, both the tags and the workflows repository are growing at a constant rate. To address this issue, we periodically (the first day of each monthïijĽuse the snapshot of the workflow repository to calculate a new tag vocabulary T and workflow collection D in n dimensions with

$$n = |T|$$

. As shown in Figure 4.3(a), the workflow w_3 is a composite workflow that contains a subworkflow inside it with the workflows w_4, w_5 and w_6. Further the workflow w_5 is a composite workflow that contains another subworkflow inside it with the workflows w_7, w_8 and w_9. So, when finding a semantically similar workflow, we consider not only the tags associated with the incomplete workflow, but also all the subworkflows inside the incomplete workflow (recursively). Let us suppose the following tag assignments are done to the workflows (w_3, w_5, w_7, w_9, wf_1, wf_2) as shown in Figure 4.3(a) 4.3(b) 4.3(c).

As shown in TABLE 4.4, we compute the tf, the idf and the tf-idf value for all the tag assignments. Then, we translate the incomplete workflow and the workflows in the suggested workflow candidates S into the corresponding vector representation. For example, the vector representation

Algorithm1: Syntactic Recommender

```
1: function syntacticRecommender
2:  input: incomplete workflow wi, list of workflows
        in repository Lw
3: output: list of syntactic recommended workflows
4: listOfProducerWorklows (LPW) ← []
5: listOfConsumerWorklows (LCW) ← []
6: for each logical port p in wi
7   if (p ∈ IP ∈ wi)
8:    for each workflow w ∈ Lw
9:      if (p is type-compatible with at least one
        logical port p* ∈ OP ∈ w)
10:       LPW ← LPW + w
11:     end if
12:   end for
13:  end if
14:  if (p ∈ OP ∈ wi)
15:    for each workflow w ∈ Lw
16:      if (p is type-compatible with at least one
        logical port p* ∈ IP ∈ w)
17:        LCW ← LCW + w
18:      end if
19:    end for
20:  end if
21: end for
22: return LPW, LCW
23: end function
```

Figure 4.4: Syntactic Recommender Algorithm.

Table 4.4: A summary of tag assignment.

TAS	TF	IDF	TF-IDF
(u_1,w_3,t_1)	1	0.54	0.54
(u_1,w_3,t_5)	1	0.84	0.84
(u_2,w_5,t_2)	0.75	0.28	0.21
(u_2,w_5,t_1)	1	0.54	0.54
(u_3,w_7,t_3)	0.75	0.84	0.63
(u_4,w_9,t_5)	1	0.84	0.84
(u_2,w_1,t_1)	1	0.54	0.54
(u_3,w_1,t_2)	1	0.28	0.28
(u_4,w_1,t_6)	1	1.14	1.14
(u_2,w_1,t_6)	1	1.14	1.14
(u_1,w_2,t_1)	1	0.54	0.54
(u_1,w_2,t_2)	1	0.28	0.28
(u_3,w_2,t_3)	1	0.84	0.84
(u_1,w_2,t_4)	1	1.14	1.14

of the incomplete workflow w_3 and the recommended workflow candidates wf_1 and wf_2 are:

Vector (wf_1) = (0.54, 0.28,1.14,1.14)

Vector (wf_2) = (0.54,0.28,0.84,1.14)

We collect all the user profiles that contain all the tags annotated by the user and translate the user profiles into the corresponding vector representations. For example, the vector representation of the user profile generated for the user u_1 (w_3, wf_2), u_2 (w_5, wf_1), u_3 (w_7, wf_2), u_4 (w_9) are:

Vector (u_1) = (0.54,0.84,0.21,0.54,0.63,0.84,0.54, 0.28,0.84,1.14)

Vector (u_2) = (0.21,0.54,0.63,0.84,0.54,0.28,1.14,1.14)

Vector (u_3) = (0.63,0.54,0.28,0.84,1.14)

Vector (u_4) = (0.84)

The workflow similarity between the incomplete workflow w_3 and the workflow candidates wf_1 and wf_2 is computed by using the corresponding vector as:

WS (w_3, wf_1) = 0.514 WS (w_3, wf_2) = 0.548

The user interest matching score between the user designing the workflow (u_1) and the workflow candidates is computed by using the corresponding vector as:

IM (u_1, wf_1) = 0.366 IM (u_1, wf_2) = 0.390

Intuitively, it is evident that, the workflow wf_2 is relevant to the incomplete workflow w3 and preferred by the user u_1 than the workflow wf_1. The final recommendation score validates that the consumer workflow candidate wf_2 (0.47) is higher than the workflow candidate wf_1 (0.44) and the threshold value is (0.45). So, we recommend the user, wf_2 as the consumer workflow to be connected to the output port of the incomplete workflow w_3.

4.5 Implementation and Case Study

We implemented the proposed folksonomy based social recommendation framework as a Web-based application called DATAVIEW, written in Java. As part of our implementation, we deployed our DATAVIEW in Futuregrids Openstack platform. We validated our proposed Syntactic Recommender and Semantic Recommender algorithms by designing a workflow downloaded from the myExperiment website.

The study focuses on workflows designed in Taverna, an open source popular workflow system. We downloaded the workflows, the input and output port details, the number of workflow instances, the user profile information and the tag assignments available in myExperiment workflow repository. The myExperiment website allows its users to share the workflows from several domains. Based on our analysis on the myExperiment dataset, as shown in Figure 4.6(a), we found that there are 9886 users, 3542 workflows, 2664 publicly available workflows and 9624 tag as-signments in the myExperiment website.

We developed a scientific workflow analyzing metabolite pathway. As shown in Figure 4.6(b), we designed the workflow based on the taverna workflow downloaded from the myExperiment website (see Figure 4.6(a)) using our data collection technique. Our workflow takes as input, the search keyword and then searches for metabolomic pathways that match the entered keywords and returns information about the chosen pathway. Although there are different ways of parallelizing scientific workflow [29], in our system, we parallelized the execution of the scientific workflow based on the number of workflow fragments in the workflow specification. A workflow fragment is a sequence of workflows that contains a source, a destination and a data channel representing the connectivity

Algorithm2: Semantic Recommender

1: **function** semanticRecommender
2: **input**: incomplete workflow wf_I, user u_m,
list of recommended producer workflows Lrpw, list of
recommended consumer workflows *Lrcw,*
recommendation weight factor γ, threshold value T
3: **output**: list of semantic recommended workflows
4 *listOfProducerWorklows (LPW)* ← []
5: *listOfConsumerWorklows (LCW)* ← []
6: *LPW_score=0*
7: *LCW_score=0*
8: **for** each workflow $wf_j \in Lrpw$
9: $LPW\_score = \gamma \cdot WS(wf_i, wf_j) + (1-\gamma) \cdot IM(u_m, wf_j)$
10: **if** (*LPW_score* > *T*)
11: *LPW* ← *LPW* + w
12: **end if**
13: **end for**
14: **for** each workflow $wf_j \in Lrcw$
15: $LCW\_score = \gamma \cdot WS(wf_i, wf_j) + (1-\gamma) \cdot IM(u_m, wf_j)$
16: **if** (*LCW_score* > *T*)
17: *LCW* ← *LCW* + w
18: **end if**
19: **end for**
20: **return** *LPW, LCW*
21: **end function**

Figure 4.5: Semantic Recommender Algorithm.

(data flow) between the workflows. We identified all the independent workflow fragments in the workflow specification and captured the workflows that are part of those workflow fragments. Then, we executed each workflow fragment separately by running them in different virtual machines.

As shown in Figure 4.6(b), our workflow contains eight primitive workflows and is deployed using four virtual machines. The input data set, keyword of data type String is sent to the virtual machine VM_1, data is processed using the Search_for_pathway, Extract_pathway_data and Choose_id workflows and the output data set generated is sent to VM_2, VM_3 and VM_4 respectively. In virtual machine VM_2, the input data is processed by Fetch_pathway_image workflow and generates an image output of type file. In virtual machine VM_3, the input data is processed by an incomplete Fetch_pathway_description workflow, whose output port is of type string and is not connected to any consumer workflow or output data stub. In virtual machine VM_4, the input data is processed by Fetch_compounds, Extract_compound_data and Fetch_compound_description workflows and generates two outputs, compound_ids of type list<Integer>, compound_infos of type list<String>. The in-progress workflow contains one incomplete workflow and we provided our users with a recommendation link. When our users, request for recommendation by clicking on the link, we recommended a list of suitable consumer workflows that shall be connected to the output port of the Fetch_pathway_description workflow. First, we identified the list of syntactically compatible workflows by executing our syntactic workflow recommender to get a list of workflows that contains at least one input port of type string and formulate the list of syntactically compatible workflows. Second, we identified the list of semantically compatible workflows by filtering the list

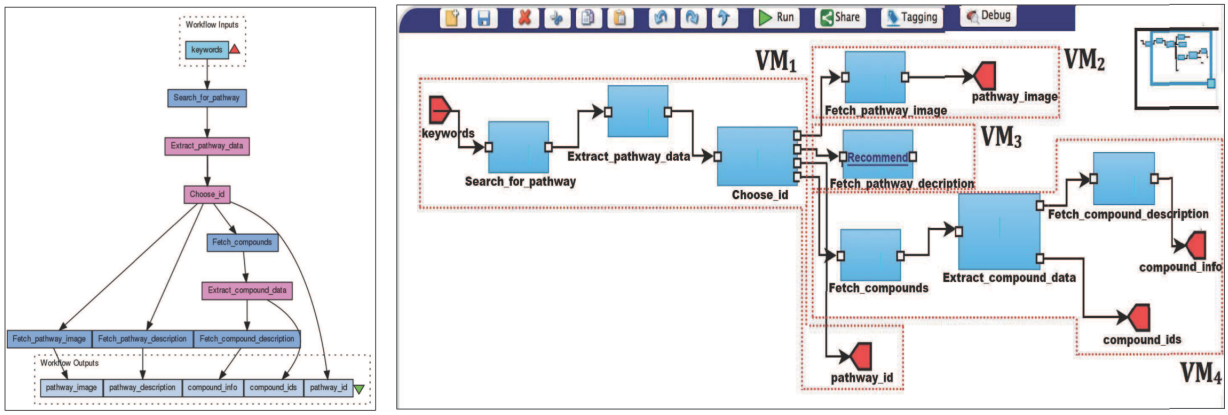


Figure 4.6: Analyzing metabolite pathway workflow.

generated in the previous step and executed our semantic workflow recommender to formulate the list of workflows that are both relevant to the incomplete workflow and also preferred by the user who designed the in-progress workflow. In Figure 4.7(b), we show the recommendation scores of the top 10 recommended workflows that are suitable to be connected to the output port of the incomplete Fetch_pathway_description workflow. In our experiment setting, we set the threshold value to be 0.5 and the recommendation weight factor

$$\gamma = 0.5$$

4.6 Chapter Summary

In this chapter we proposed a folksonomy-based social workflow recommendation system to improve workflow design productivity. We discussed the details about our web-based workflow design environment (called Web-bench) to allow users to create workflows and collaboratively annotate and categorize them using social tags. The resulted folksonomy improves workflow searchability and shareability. We proposed several workflow recommendation strategies to automatically or semi-automatically augment an in-progress workflow, leveraging both structural and semantic similarities between workflows and guiding information extracted from previously created workflows in the database. We discussed the details on how we implemented the proposed environment and strategies in a prototype based on the DATAVIEW scientific workflow management system and validated our approach with numerous use cases.

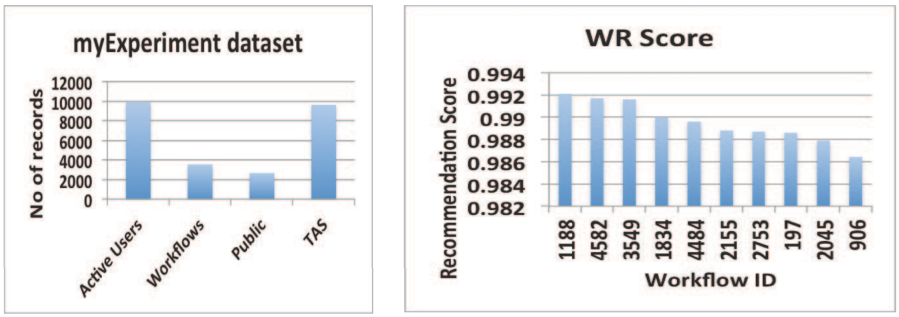


Figure 4.7: (a) myExperiment dataset; (b) Workflow recommendation score.

CHAPTER 5: A NOSQL COLLECTIONAL DATA MODEL FOR RUNNING BIG DATA WORKFLOWS

5.1 Introduction

Big data workflows have recently emerged as the next generation of data-centric workflow technologies to address the five "V" challenges of big data [**?**]: volume, variety, velocity, veracity, and value [**?**, 46, 47, 49]. While its precedent, scientific workflows, focus on dataflow and automation management [**?**], big data workflows focus on large-scale data processing and analytics with a "scale-out" architecture and a "moving-computation-to-data" processing paradigm. More formally, a big data workflow is the computerized modeling and automation of a process consisting of a set of computational tasks and their data interdependencies to process and analyze data of ever increasing in scale, complexity, and rate of acquisition. The coining of the term "big data workflows" is timely and important to recognize the continuing relevance and importance of workflow technologies in data processing and management, as well as the challenges and opportunities of big data research in the workflow community [63]. While more and more big data tools have been developed in recent years to address the big data deluge in both science [46] and business [47], the gap between the capability of data collecting and the power of data processing and analysis continues to increase. One category of such tools are NoSQL databases [48], which deliver high read and write performance by automating the data distribution and retrieval over a cluster of tens of thousands of machines [50]. In contrast to their SQL counterpart, NoSQL databases often relax the traditional ACID properties of transactions and introduce restrictions on its query language, such as no-support-for-join, in favor of high performance of read and write. The wide adoption of NoSQL techniques in big data applications is attributed to, among other things, their large scalability, high fault-tolerance, flexible data models, and high performance query capability [50].

However, the power of big data not only lies in storing and querying large datasets, but also in performing efficient ad hoc sophisticated analysis over such datasets to shorten the cycle of "from data to insight and to value". One major research question is: Is it possible to leverage the power of NoSQL techniques in big data workflow systems to improve the performance of workflow execution? If yes, how? One approach is to integrate an existing NoSQL database system into a big data workflow system. This approach, however, will not unleash the full power of neither as data move-

ment between the NoSQL database and the workflow engine will become the bottleneck. Moreover, many of the workflow optimization opportunities will not become available under the constraints of a NoSQL database, which decides the placement of data according to partitioning strategies that are optimized for querying, not for ad hoc analysis, in which data placement, replication, and data movement need to be decided on the fly according to the structure and data access patterns of a workflow [65,66]. Therefore, we take another approach in this research, in which we develop our own NoSQL collectional data model, which leverages some of the capabilities of existing NoSQL data models, while enriching in capabilities, such as flexible MapReduce workflows, workflow executors, and optimization of workflow execution.

In this chapter, we propose a NoSQL data model that: 1) supports high-performance MapReduce-style workflows that automate data partitioning and data-parallelism in workflow execution; in contrast to the traditional MapReduce framework, our MapReduce-style workflows are fully composable with other workflows, and thus enable dataflow applications with a richer structure; 2) automates virtual machine provisioning and deprovisioning on demand according to the sizes of input datasets; 3) enables a flexible framework for workflow executors that take advantage of the proposed NoSQL data model to improve the performance of workflow execution. Our case studies and experiments show the competitive advantages of our proposed data model. The proposed NoSQL data model is implemented in a new release of DATAVIEW, one of the most usable big data workflow systems in the community.

5.2 An Overview Of DATAVIEW

In Figure 5.1, we present the overall architecture of DATAVIEW, which enriches the original architecture described in [63] with a refined design for the Workflow Engine. DATAVIEW consists of seven subsystems: the Webench is a Web-based interface that supports user interaction, workflow visualization, data presentation, and system configuration. The Data Product Manager features the proposed NoSQL collectional data model and a rich set of other data types, including relational, files, and scalar types. The Task Manager supports a single-component based task model that separates registration from configuration and eases the process of registering external functional components (such as Web services) into primitive workflows [59]. The Task Manager is also responsible for the runtime execution of primitive workflows. The Cloud Resource Manager (CRM) provides the

provisioning and deprovisioning capabilities of cloud resources, including both virtual machines and storage resources. The Provenance Manager manages the data lineage and derivation history of data products for the reproducibility and validation of workflow execution results [44]. The Workflow Monitor supports the monitoring of workflow execution status and progress and exception handling [45]. Finally, the Workflow Engine is at the heart of the DATAVIEW system, responsible for overall workflow orchestration, scheduling, and the coordination and collaboration of all subsystems. The Workflow Engine is enriched with the notion of "executors", which abstracts the execution platform that a workflow will be executed at runtime. Two categories of workflow executors are supported: on-premises workflow executor, which supports the execution of a workflow on a single DATAVIEW server, and cloud workflow executor, which supports the execution of a workflow in the cloud (e.g., Amazon EC2). Moreover, two types of cloud workflow executors have been implemented: type-A cloud workflow executor supports a clustering algorithm that partitions a workflow into a number of workflow clusters, with each workflow cluster executed in one virtual machine; type-B cloud workflow executor supports MapReduce-style workflows, which automates data partitioning, virtual machine provisioning and deprovisioning, and scalable execution of workflows. Both type-A and type-B workflow executors exploit the proposed NoSQL collectional model for improved performance of workflow execution.

5.3 NoSQL Collectional Data Model

A big data workflow represents a multiple-step data analysis pipeline by chaining several data analysis modules together via data links that connect the output of one analysis module to the input of another analysis module. A big data focuses on large-scale data processing and analytics with a "scale-out" architecture and a "moving-computation-to-data" processing paradigm. Big data imposes challenges in the workflow development at both the primitive and composite workflow level. A primitive workflow is the workflow that contains no sub-workflows in it. On the other hand, a composite workflow has one or more sub-workflows inside it. Our original collectional data model [30] is a counterpart of relational data model that supports creation of hierarchically organized data in a nested manner. In our new NoSQL collectional data model, we extend our original collectional data model to improve the performance of big data workflow execution.

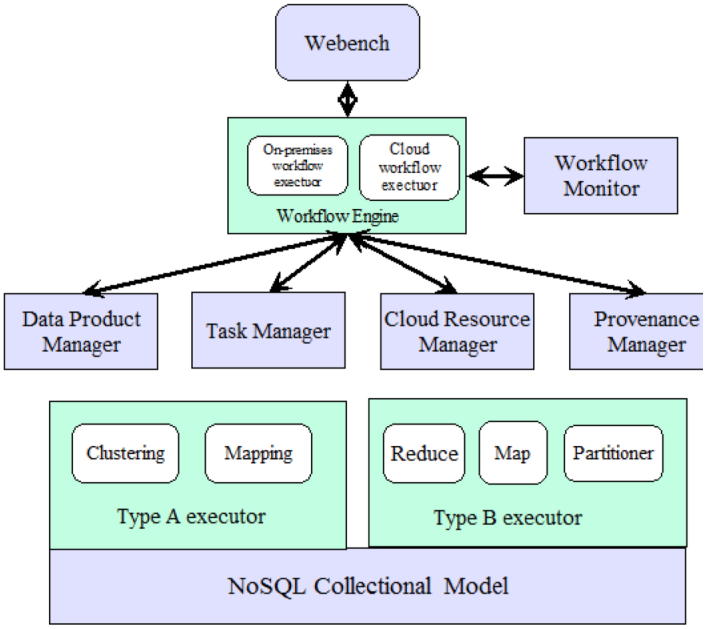


Figure 5.1: Architecture of DATAVIEW

Big data workflow is executed in the Cloud. A cloud consists of a set of virtual machines that are used to store the partitioned input data, execute the workflow and store the output data generated by the workflow. A big data workflow is defined as follows:

Definition 5.1: A big data workflow w is a 8-tuple (IP, OP, D, T, S, Consumer, Producer, DataType), where

- IP is the set of input ports for workflow w. Each individual input port is denoted by ip_m, IP=$\{ip_1,ip_2,...,ip_M\}$. $\hat{IP}$ is the set of intermediate input ports for workflow w and $\hat{IP} \subset$ IP.
- OP is the set of output ports for workflow w. Each individual output port is denoted by op_q, OP=$\{op_1,op_2,...,op_Q\}$.
- D is the set of workflow datasets. Each individual dataset is denoted by d_j, D=$\{d_1,d_2,d_3,...,d_J\}$. d_j can be either connected to $ip_m \in$ IP or $op_q \in$ OP.
- T is the set of workflow tasks. Each individual task is denoted by t_i, T=$\{t_1,t_2,t_3,...,t_I\}$. Each task can have one or more input and output ports. $t_i.\ ip_m$ shows the mth input port of task t_i. subsequently $t_i.\ op_q$ shows the qth output port of task t_i.

- S:D $\rightarrow R^+$ is the dataset size function. S(d_j), $d_j \in$ D returns the size of the dataset d_j. The size of a dataset is defined in some pre-determined unit such as MegaBytes, GigaBytes, TeraBytes, etc. R^+ is the set of positive real number.

- Consumer:T $\rightarrow 2^{\hat{}}T$ is the task-task function. Consumer(t_i), $t_i \in$ T returns the set of tasks that t_i is directly connected to and require the output of t_i for their inputs.

- Producer:T $\rightarrow 2^{\hat{}}T$ is the task-task function. Producer(t_i), $t_i \in$ T returns the set of tasks that are connected to t_i directly and t_i require their output as its inputs.

- DataType:D $\rightarrow$ {"Scalar", "File", "Relational", "Collectional"} is the data-type function. DataType(d_j) returns the type of data, d_j

Our NoSQL collectional data model supports registering hierarchically organized and collection oriented datasets. We formally define a NoSQL Collectional Data Model as follows:

Definition 5.2: A NoSQL Collectional Data Model A NCDM of level K can be formalized by C = ([C_1, C_2,..., C_K], V, R, I, K), where:

- I and K are two positive integers and $1 \leq I \leq K$.

- The list [C_1, C_2,..., C_K] is called as the primary key of the collection; therefore functional dependency (C_1, C_2,..., C_K) $\rightarrow$ V holds.

- A prefix [C_1, C_2,..., C_I] of [C_1, C_2,..., C_K] is called a partition key of G, all tuples that correspond to the same value of [C_1, C_2,..., C_I] will be mapped to the same virtual machine in physical organization.

- V is an attribute for the value, it can take the type of a scalar value (INTEGER, STRING, FLOAT, DOUBLE, RELATIONNAME) or a relational name of schema specified by R. R is a relational schema.

In Figure 5.2 we show an example of OpenXC collectional data set from the automotive domain. It consists of three key attributes, drivers, vehicles and traces, and value is a relational name with three attributes <Name, Timestamp, Value>. In Figure 5.3 we show the OpenXC collectional data set that is partitioned and stored in three virtual machines, vm_1-vm_3.

Definition 5.3: A Data Partitioner γ is used to partition a collectional instance c of schema $C = ([C_1, C_2,..., C_K], V, R, I, K)$ into $c_1, c_2,..., c_M$ such that $c_1 \cup c_2 \cup ... \cup c_M = c$, where c_m is the partition for virtual machine m. Let α: $C_1 \times C_2 \times ... \times C_I \rightarrow$ [-263, 263-1] be a function that computes the token for a given collectional tuple t of c using only the value of the partition key (possibly composite). Let β: [-263, 263-1] $\rightarrow$ [1, M], then we have $\gamma(t) = \beta(\alpha(\pi(t,I)))$ where $\pi(t,I)$ is the projection of t over the first I attributes.

Our proposed Map construct extends our previous notion of the Map workflow construct in [32] in two directions: 1) our Map constructs abstracts a data partitioner γ implicitly where the primary key of c is used as the partition key; 2) our Map constructs supports fully the NoSQL collectional data model, and thus fully exploits the data parallelism and the dynamic resource provisioning capability of our cloud resource manager.

Definition 5.4: The Map construct abstracts the partitioning and distribution of a large collectional data product over a set of M virtual machines for high-performance parallel processing. Given a workflow $w([i_1, i_2, ... , i_n], o)$ with n input ports and one output port, where i_1 takes a collectional data product as its input, we have $Map(w)(c, i_2, ... , i_n) = \bigcup_{m=1}^{M} w(c_m, i_2, ... , i_n)$. That is, the output of Map(w) on collectional data product c is equal to the union of the application of w on each partition c_m. In order to apply Map on w, w must satisfy the following constraints:

1. i_1 takes a collectional data product of schema $C = ([C_1, C_2,..., C_K], V, R, I, K)$ as input.
2. w is a primitive workflow or a composite workflow with no nesting Map/Reduce constructs.
3. $w(c, i_2, ... , i_n) = \bigcup_{tec} w(t, i_2, ... , i_n)$. That is, the output of w on input collection c is equal to the union of the application of w on each tuple in c.

Our proposed Reduce construct extends our previous notion of the Reduce workflow construct in [32] in two directions: 1) automatic shuffling and redistribution of the input collectional dataset into multiple virtual machines; 2) executing the workflow that performs aggregation on the input collectional dataset based on the user provided key attribute in multiple virtual machines in parallel.

Definition 5.5: The Reduce construct abstracts the automatic shuffling, redistribution, aggregation of a large collectional data product based on a given key attribute C_I over a set of

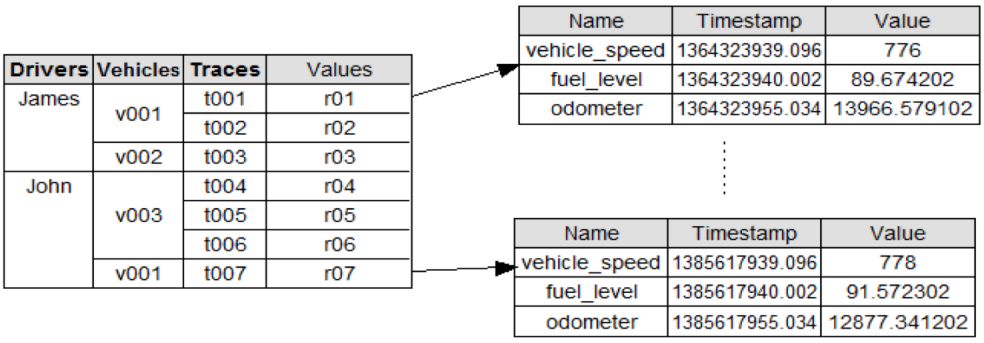


Figure 5.2: OpenXC collectional data product.

M virtual machines for high-performance parallel processing. Given a workflow w([i_1, i_2,..., i_n], o) with n input ports and one output port, where i_1 takes a collectional data product as its input, we have Reduce(w, C_I)(c, i_2,..., i_n) = $\bigcup_{m=1}^{M}$ w(c_m, i_2,..., i_n). That is, the output of Reduce(w, C_I) on collectional data product c is equal to the union of the application of w on each group c_m. In each group c_m, for t_1, $t_2 \in c_m$ we have $\pi(t_1,C_I) = \pi(t_2,C_I)$. That is, all tuples in c_m have the same value for C_I.

Note that in the Map construct, a Map workflow run is applied to each tuple in c, while in the Reduce construct, a Reduce workflow run is applied to a group of tuples in c, with each group shares the same value for the given attribute C_I. Therefore, the Reduce construct is to be applied to a workflow that implements an aggregation function that is applicable to each group of the given input collectional data product as reshuffled according to the given key attribute.

A workflow executor takes a big data workflow, provisions virtual machines in the cloud, partitions the input collectional data product, executes the tasks of the workflow on different virtual machines, and finally deprovisions the assigned virtual machines and presents the output of the workflow to the user. While type-A workflow executor is used to execute a graph-based workflow, individual Map/Reduce workflow tasks are executed by the type-B workflow executor. We present the algorithms for type-A and type-B workflow execution in the next section.

5.4 Algorithm For Workflow Executors

In this section, we propose three new algorithms that are implemented in our cloud workflow executor. We automatically provision and deprovision virtual machines based on both the size of input datasets connected to the workflow and the structure of the workflow.

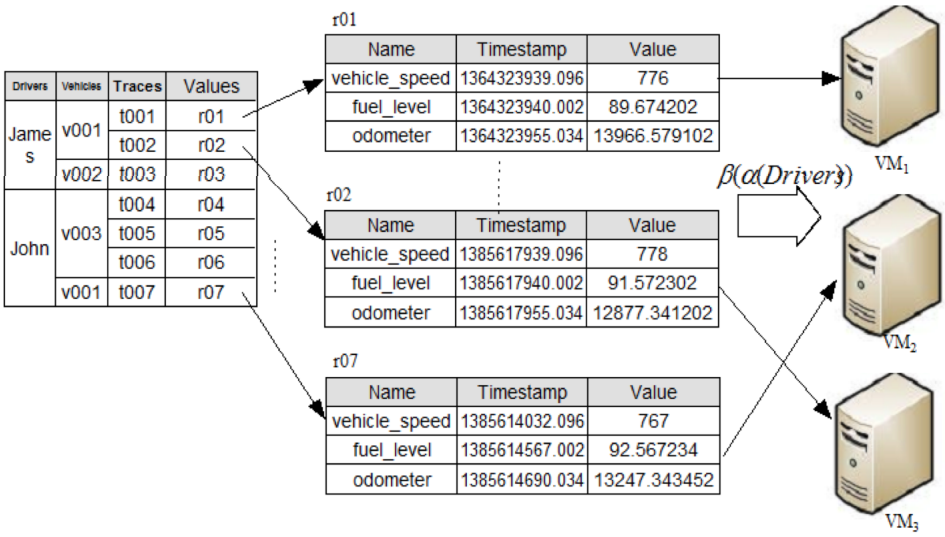


Figure 5.3: Example of OpenXC data partitioner.

We provide two types of parallelism. First, we provide a workflow level parallelism, type-A, by leveraging the structure of the workflow, we cluster the given workflow into multiple workflow clusters. Each workflow cluster consists of a set of tasks that are executed in the same virtual machine. Different workflow clusters are executed in different virtual machines in parallel.

Second, we provide a task level parallelism, type-B, by leveraging our newly proposed NoSQL collectional data model. We automatically partition the input datasets into multiple virtual machines and the task is mapped to those machines and executed in parallel. We demonstrate Algorithm 1, 2 and 3 by using the example shown in Figure 5.4.

5.4.1 Task Clustering

The goal of the Task Clustering Algorithm (T-Cluster) is to generate a cluster map for a given workflow and automatically provision and deprovision virtual machines. The cluster map consists of a list of pairs $<t_i$, VMID$>$, such that t_i is the name of the task and VMID is the identifier of a virtual machine. Each task in a workflow consists of a set of producers and a set of consumers that are connected to it, except for the entry and exit tasks. The entry tasks in the workflow do not contain any producer and instead a set of input datasets are connected to it. In the same manner, the exit tasks do not contain any consumer and instead a set of output stubs are connected to it, in order to visualize the final results of the workflow.

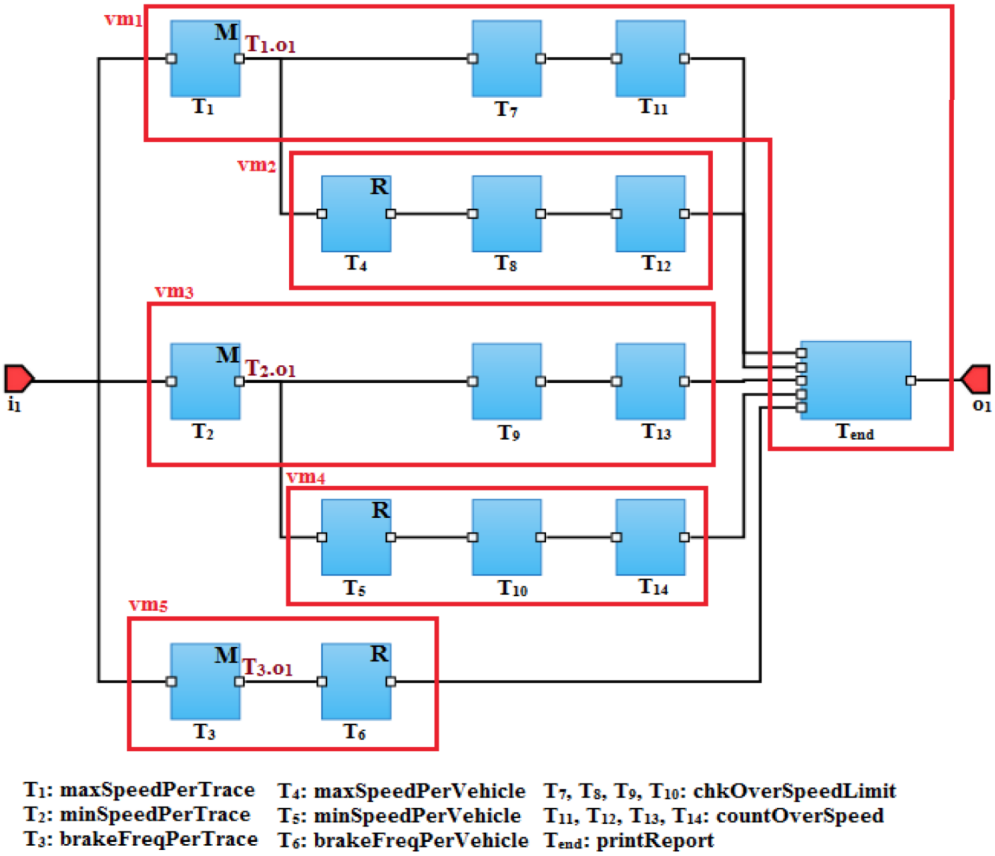


Figure 5.4: An example big data workflow.

In Figure 5.4, we show an example big data workflow w from the automotive domain that is used to query the OpenXC dataset and compute the speeding and braking behavior of the driver. In Algorithm1, in line 4, we get all the tasks in w in some topological order and assign it to list ts. We validate the correctness of our algorithm for different topological order of ts. Let ts = <T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14}, T_{end}>. In line 5, we iterate through each task t_i in ts and add t_i to the cluster map d. All the entry tasks in w are added to list en = <T_1, T_2, T_3>. Between lines 8 and 30, we iterate through each task t_i in ts and add the consumer of t_i to list cs. For example consumers of T_1 are <T_4, T_7>. If task t_i is an element of list en, then we assign VMID to task t_i and increment VMID.

For example, <T_1,1> is added to the map since the default value of VMID = 1. If task t_i is not an element of list en, then we iterate through each consumer c in cs. For the first consumer, we assign the same VMID as that of the task and for the other consumers, we assign a different VMID. For example, for the consumers of T_1, the following pairs are added, <T_4,1>, <T_7,2>. VMID is incremented every time a new pair is added to the map. The final map d = <(T_1, 1), (T_2,3), (T_3,5), (T_4,2), (T_5,4), (T_6,5), (T_7,1), (T_8,2), (T_9,3), (T_{10},4), (T_{11},1), (T_{12},2), (T_{13},3), (T_{14},4), (T_{end},1)> is returned as an output for workflow w. The output map d of w consists of five clusters assigned to five virtual machines.

5.4.2 Type-A Cloud Workflow Executor

The goal of the type-A workflow executor is to reduce the workflow makespan for a given workflow by running each cluster of the workflow in parallel in multiple virtual machines in the cloud. In Algorithm 2, in line 4, we get the cluster map d from Algorithm1(T-Cluster) for the workflow w. In line 5, we get all the tasks in workflow w in some topological order and store it in list ts. Between lines 6 and 24, we iterate through all task t_i in ts in parallel. We use inputready to determine whether all the datasets are available in order to execute a task. In line 7, we set inputready[t_i] to the total number of non-intermediate input ports of t_i. In Figure 5.4, for example inputready[T_1] = 1, inputready[T_4] = 0.

At each iteration, every task except the entry tasks will wait until all the input datasets for the task becomes ready. For example, T_{end} will continue to wait until the tasks T_{11}, T_{12}, T_{13}, T_{14}, T_6 are executed and the data movement from T_{11}, T_{12}, T_{13}, T_{14}, T_6 to T_{end} is completed.

Between lines 13 and 23, we iterate through each consumer c_i of task t_i in parallel. If the consumer c_i and task t_i are assigned to different virtual machines, then we move the data from $d[t_i]$ to $d[c_i]$. We increment $inputready[c_i]$ through a locking mechanism so that at a particular time only one consumer of one task can increment it. For each consumer c_i, the number of input ports is calculated. If $inputready[c_i]$ is equal to the total number of input ports of c_i, then we send a signal to the consumer c_i as a wake up call. At that point $inputready[c_i]$ is validated to check whether all the datasets needed to execute c_i is ready. If the datasets are ready, then c_i is executed and the consumers of c_i are processed. For example only after execution of the tasks T_{11}, T_{12}, T_{13}, T_{14}, T_6, the signal to wake up T_{end} is sent from T_{14}.

5.4.3 Type-B Cloud Workflow Executor

The goal of the type-B workflow executor is to reduce the task makespan for any task that has the Map or the Reduce construct applied on it. A construct is a high order function that is used to transform any given function into another sophisticated function. In our case, a construct is applied on a task to transform the task into a Map or a Reduce task. The Map construct is applied to a task that performs extracting, filtering and transformation. The Reduce construct is applied to a task that performs aggregation, summarization, filtering and transformation. Besides, the Map and Reduce constructs also differ in the way the input dataset is partitioned. Our data partitioner inspired from Cassandra uses our custom hash function, to distribute any given NoSQL collectional dataset into multiple virtual machines. Our cloud infrastructure manages a ring with a range for each virtual machine. We assign multiple tokens for each range. The advantage foreseen in our approach is that we support the dynamic addition and delete of virtual machines and still manage to keep our key to token mapping information intact. We plan to discuss our partitioner as a separate research article.

We apply the Map construct on the tasks T_1, T_2, T_3 and the Reduce construct on tasks T_4, T_5, T_6 on the big data workflow w shown in Figure 5.4. In Algorithm 3, in line 4, we initialize out, z, n and vms. In line 5, we get all the input datasets connected to task t_i and store them in in. Between lines 6 and 21, we iterate through all the datasets d in in. In line 8, we validate whether the type of the dataset d is collectional. On validation success, we compute the total number of virtual machines dynamically based on the size of the dataset and the user configured value for partition

Algorithm1: Task Clustering

```
1:  function T-Cluster
2:  input: workflow specification w
3:  output: d, a map storing task-VM assignments.
4:  ts ← all tasks in w sorted in a topological order
5:  for each t ∈ ts do d[t] ← 0 end for
6:  en ← all entry tasks of w
7:  VMID = 1, fc = false
8:  for each t ∈ ts
9:    cs ← all consumers of t
10:   if (t ∈ en)
11:     d[t] ← VMID
12:     for each c ∈ cs
13:       if (d[c] = 0)
14:         d[c] ← VMID
15:         VMID ← VMID + 1
16:       end if
17:     end for
18:   else
19:     fc = true
20:     for each c ∈ cs
21:       if (d[c] = 0)
22:         if (fc = true)
23:           d[c] ← d[t]
24:           fc = false
25:         else
26:           d[c] ← VMID
27:           VMID = VMID + 1
28:         end if
29:       end if
30:     end for
31:   end if
32: end for
33: return d
34: end function
```

Figure 5.5: Task clustering.

Algorithm2: Type-A workflow executor

```
1: function Type-A
2: input: workflow specification w
3: output: exit code
4: d ← T-Cluster(w)
5: ts ← all tasks in w sorted in a topological order
6: forall task t_i ∈ ts in parallel
7:   inputready[t_i] = |t_i.IP|- |t_i.IP^|
8:   while (inputready[t_i] < | t_i.IP|)
9:     wait(sig[t_i]);
10: end while
11: execute t_i on d[t_i]
12: cs ← all consumers of t_i
13: forall consumer c_i ∈ cs in parallel
14:     if (d[t_i] ≠ d[c_i])
15:       Move OutputOf (t_i, c_i) from d[t_i] to d[c_i]
16:     end if
17:     lock → acquire()
18:       inputready[c_i] = inputready[c_i] + 1
19:     lock → release()
20:     if (inputready[c_i] = |c_i.IP|)
21:       signal(sig[c_i]);
22:     end if
23: end in parallel
24: end in parallel
25: return SUCCESS
26: end function
```

Figure 5.6: Type-A workflow executor.

Algorithm3: Type-B workflow executor

```
1: function type-B
2: input: task name t_i, type of construct toc,
   partition size ps, number of task runs per
   virtual machine RunsPerVM, partition key
   for reduce r_k
3: output: final output of task t
4: out ← [], vms ← [], z = 0, n = 0
5: in ← t_i.inputdatasets;
6: for each dataset d ∈ in
7:   if (Type (d) = collectional)
8:     z = size of d
9:     n = z / ps / RunsPerVM
10:    vms ← CRM.provision(n)
11:    if (toc = map)
12:       ParKeys ← get all keys of d
13:    else if (toc = reduce)
14:       ParKeys ← r_k
15:    end if
16:    Partition(d, ParKeys) to vms
17:  else
18:    Move(d) to vms
19:  end if
20: end for
21: forall vm ∈ vms in parallel
22:  result ← Execute task t_i in vm
23:  out ← out ∪ result
24: end in parallel
25: return out
26: end function
```

Figure 5.7: Type-B workflow executor.

size (ps) and total number of task runs per virtual machine (RunsPerVM). In line 11, we provision the virtual machines. In line 12, we validate the type of construct applied to task t_i. If the type of the construct is a Map, then we partition the data with all the key attributes in dataset d. If the type of the construct is Reduce, then we partition the data by a user provided key attribute r_k. For all other datasets connected to task t_i that are not of type collectional, we move the original dataset to all the virtual machines. Between lines 22 and 25, we run task runs of t_i in parallel in all the virtual machines vms. The results from all task runs are combined together and returned as the final output out.

5.5 Case Study and Experiments

In our DATAVIEW system, we implemented a big data workflow called the Autoanalytics workflow to show the strength of our proposed NoSQL collectional data model and the scalability features. The Autoanalytics workflow is used to analyze the data collected from vehicles and provide insights on the risk level based on the drivers driving behavior. OpenXC is an open source platform that is used as a source to generate a wealth of data from the vehicle through a hardware device that is installed in the car. As the average adult driver in the US may generate up to 75 Gb of such driving data annually, the total amount of data generated in the US may exceed 14 Eb (1018 bytes) per year [43, 63]. We collected data from X drivers for one hour with X= 5, 10, 15, 20, 25 resulting datasets of size in the range of 1GB-5GB. Because of the dynamic nature of the growth of size of the dataset and the 5V big data challenges incurred in the domain, we consider our Autoanalytics workflow as a big data workflow.

In Figure 5.5, we show the Autoanalytics big data workflow. The first step is *Extract-DriverDetails* that accepts the collectional OpenXC dataset as input and filter by their VehicleId. The second step is *ComputeSpeedDistribution* that is used to find the topK vehicle speed and the distance driven without pressing the brake. The third step is *AddressFinder* that is used to find the geographic location of the vehicle during the time at which the signal was captured. We use Google Places API to determine the address from the latitude and longitude signal values. The fourth step is *ComputeRoadType* that is used to find the type of the road (highway or local) for each geographic location computed in the third step. The fifth step is *SpeedLimitFinder* that is used to find the speed limit posted on those geographic locations based on the road type. The sixth

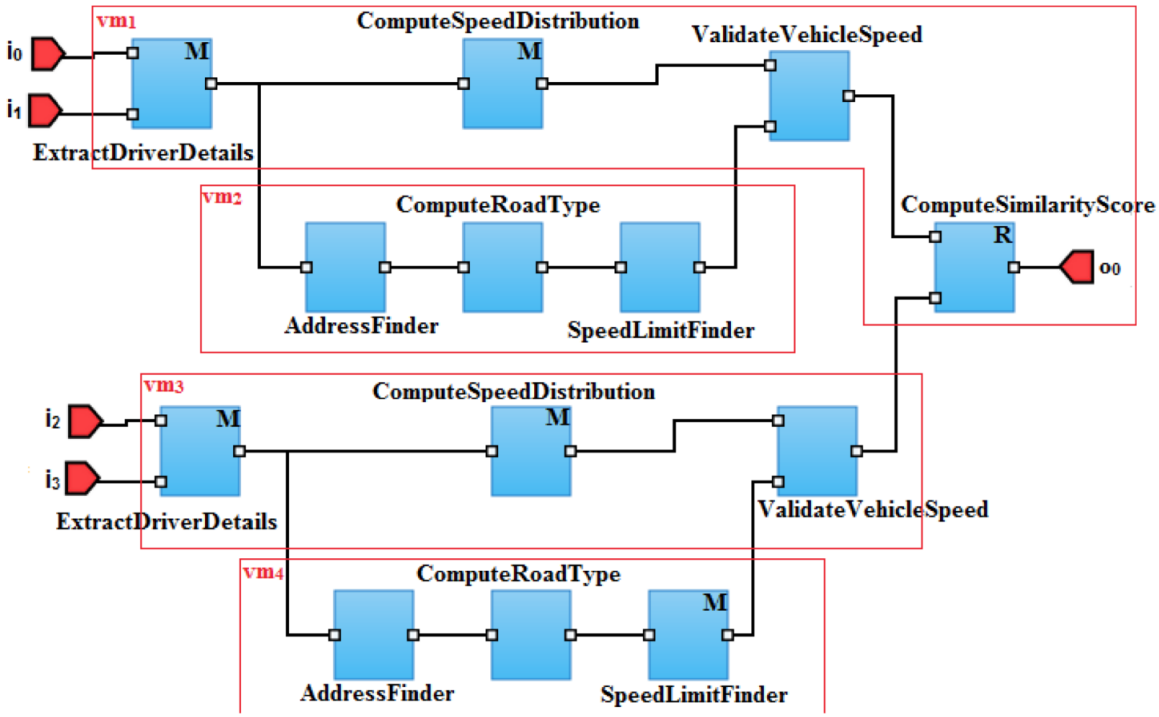


Figure 5.8: An automative OpenXC big data workflow.

step is *ValidateVehicleSpeed* that is used to compare the vehicle speed generated in step-2 with the actual speed limit in the closest latitude longitude. The output of the task indicates if the driver was below or over the speed limit. The 7th step is *ComputeSimilarityScore* that is used to compare several drivers with different traces to identify the similarity between them. The final output of the *ComputeSimilarityScore* generates a report to show the list of good drivers and bad drivers along with their driving score.

We apply the type-B Map construct on step-1, step-2 and executed them in the range of 1-25 virtual machines simultaneously. We apply the type-B Reduce construct on step-7 by (key=DriverName) to partition/group all the signal values associated with each driver into the same machine. We apply the type-A cloud workflow executor for 25 drivers and executed the workflow in the range of 1-25 virtual machines in the EC2 cloud. We performed our experiments in the OpenXC dataset with the range of 1-5GB on 1 to 25 machines. Our experimental results show that our type-A executor performed well by reducing the workflow makespan. And we applied the Map and the Reduce construct on the individual tasks in the workflow. Our type-B executor performed well by reducing the individual task makespan and as a whole also reducing the workflow

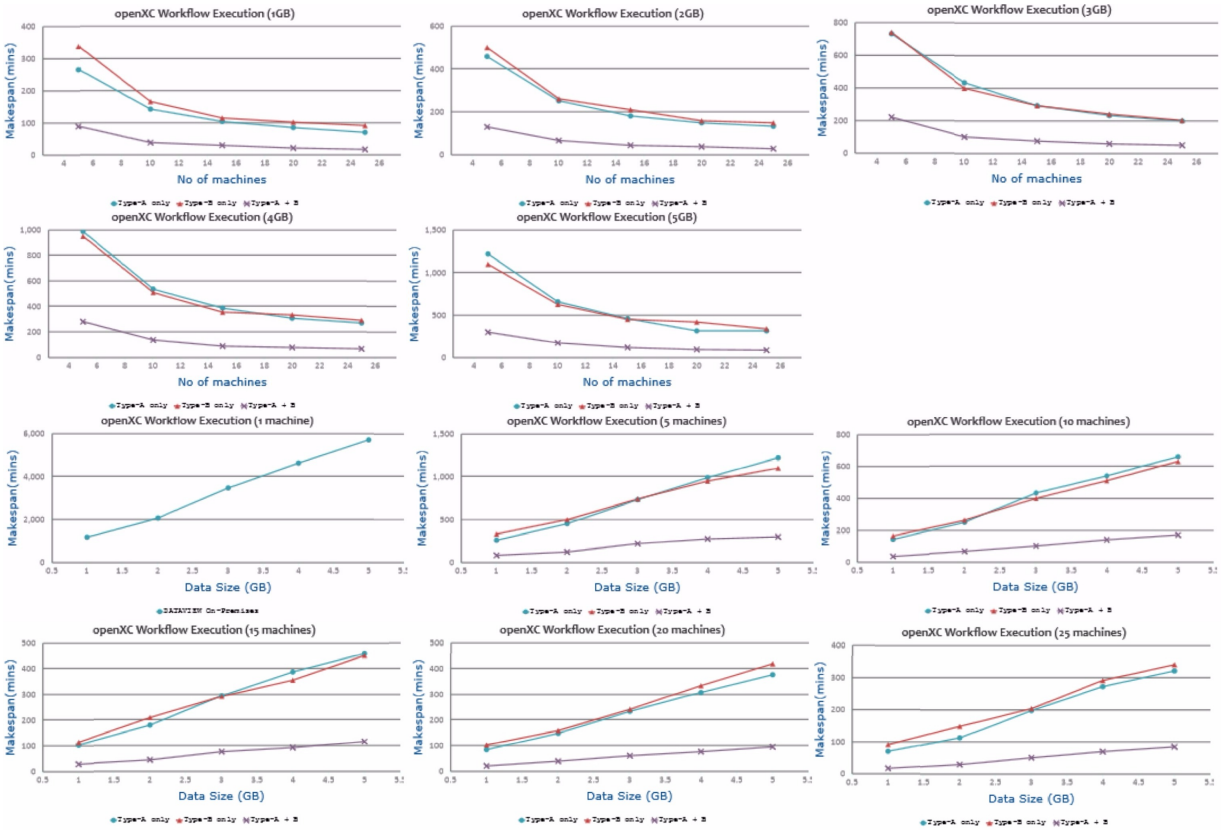


Figure 5.9: Workflow makespans by varying the number of virtual machines and datasets.

makespan even more. In Figure 5.6, we show the experimental results that was performed using the OpenXC dataset in Amazon EC2 cloud environment.

5.6 Chapter Summary

In this chapter, we proposed a NoSQL data model that: 1) supports high-performance MapReduce-style workflows that automate data partitioning and data-parallelism in workflow execution; in contrast to the traditional MapReduce framework, our MapReduce-style workflows are fully composable with other workflows, and thus enable dataflow applications with a richer structure; 2) automates virtual machine provisioning and deprovisioning on demand according to the sizes of input datasets; 3) enables a flexible framework for workflow executors that take advantage of the proposed NoSQL data model to improve the performance of workflow execution. Our case studies and experiments show the competitive advantages of our proposed data model. The proposed NoSQL data model is implemented in a new release of DATAVIEW, one of the most usable big data workflow systems in the community.

CHAPTER 6: SCHEDULING BIG DATA WORKFLOWS IN THE CLOUD UNDER BUDGET CONSTRAINTS

6.1 Introduction

Big data workflows [63] have recently emerged as a data-centric workflow approach to analyze the data that is ever increasing in scale, complexity, and rate of acquisition. Data scientists develop workflows by modeling their complex scientific applications as a set of data processing tasks with a set of data dependencies between the tasks. Although this approach facilitates the execution of tasks in a distributed cloud computing environment [67–69], it also adds a research challenge of how and where to schedule the tasks in a distributed and heterogeneous cloud environment in a usable manner [64]. The decision of how and where to schedule the tasks in a workflow is driven by the Quality of Service (QoS) requirements defined by data scientists such as the budget or the deadline for the workflow execution. The goal of the scheduling problem is to minimize the total cost for executing the workflow or makespan (i.e. total execution time of the workflow), while still meeting the QoS requirement defined by data scientists.

Cloud service providers such as Amazon EC2 offer a scalable infrastructure that allows an unlimited number of virtual machines that can be provisioned for an amount of time proportional to the cost for usage. There are different types of instances that range from less powerful to more powerful in terms of their CPU, memory, storage and networking capacity. The cost per machine usage is billed on an hourly rate with the cheapest resource offering the least performance to the most expensive resource offering the highest performance. The Amazon EC2 cloud service provider provides an easily accessible application programming interface through which the cloud resource manager of the big data workflow system can provision and deprovision resources. The primary responsibility of the workflow engine in a big data workflow system is to orchestrate the execution of the workflow by 1) identifying a type of cloud resource that is appropriate for each set of tasks in the workflow based on the QoS requirement; 2) provisioning and deprovisioning a set of resources that are of the identified resource types; 3) scheduling the tasks to the appropriate cloud resources and initiate the distributed execution process. The scheduling problem is non-trivial. In fact, it is a well known NP-complete problem [51].

In this chapter, we propose a new Big dAta woRkflow schEduler uNder budgeT conStraint known as BARENTS that supports high-performance workflow scheduling in a heterogeneous cloud computing environment with a single objective to minimize the workflow makespan under a provided budget constraint. Our case study and experiments show the competitive advantages of our proposed scheduler. The proposed BARENTS scheduler is implemented in a new release of DATAVIEW, one of the most usable big data workflow systems in the community.

6.2 Workflow Scheduler Model

A cloud computing environment provides a framework for enabling ubiquitous, on-demand access to a shared pool of resources of heterogeneous types which can be provisioned and deprovisioned with a minimal management effort. The computation tasks and their data counterpart are moved to the resources in the cloud, in order to perform the actual execution of the tasks. Every individual resource is associated with a predetermined cost, computing speed function and data communication rate function. More formally a cloud computing environment is defined as:

Definition 6.1: A cloud computing environment is a 6-tuple C(R, R_T, R_C, F_B, F_R, R_S), where

- R is a set of resources. Each individual resource is denoted by R_i in the cloud computing environment.
- R_T is a set of resource types such as {"t2.nano", "t2.micro", "t2.small", "t2.medium", "t2.large", ...}.
- R_C: $R \rightarrow Q^+$ is the resource usage cost function. $R_C(R_i)$, $R_i \in R$ gives the cost in some dollar amount for the resource usage R_i in the cloud computing environment. The resource with the minimum R_C is called $R_{cheapest}$ and the resource with the maximum R_C is called $R_{expensive}$.
- F_B: $R \times R \rightarrow Q^+0$ is the data communication rate function. $F_B(R_{i1}, R_{i2})$, $R_{i1}, R_{i2} \in R$ gives the data communication rate between R_{i1} and R_{i2}. Q^+0 is some pre-determined unit like bytes per second.

- F_R: $R \rightarrow Q^+$ is the resource computing speed function. $F_R(R_i)$, $R_i \in R$ gives the speed for the computing resource R_i, measured in some pre-determined unit like million instructions per machine cycles or million instructions per nanoseconds.

- F_S: $R_T \rightarrow R$ is the resource provisioning function. $F_S(R_t)$, $R_t \in R_T$ returns a resource instance of the resource type of R_t.

A big data workflow is the computerized modeling and automation of a process consisting of a set of computational tasks and their data interdependencies to process and analyze a large amount of data. The big data workflow consists of a set of interconnected tasks and their data counterparts. Because one or more of the input datasets connected to the tasks fall under the umbrella of big data, there is a need for a distributed computing environment such as the cloud to process the tasks in an efficient manner. More formally, a big data workflow is defined as: ***Definition 6.2:*** A big data workflow can be formally defined as a 4-tuple W = (T, D, F_T, F_D), where

- T is a set of tasks in the workflow W. Each individual task is denoted by T_k.

- D = {<T_{k1}, T_{k2}> | T_{k1}, T_{k2} $\in$ T, k1 $\neq$ k2; k1,k2 $\leq |T|$, T_{k2} consumes data $D_{k1,\,k2}$ produced by T_{k1}} is a set of data dependencies. $D_{k1,k2}$ represents an amount of data required to be transferred after T_{k1} completes and before T_{k2} starts. D_k represents all the output datasets from task T_k.

- F_T: $T \rightarrow Q^+0$ is the execution cost function. F_T (T_k); $T_k \in T$ gives the execution cost of a task T_k, measured in some pre-determined unit like million instructions per machine cycles or million instructions per nanoseconds.

- F_D: $D \rightarrow Q^+0$ is the data size function. F_D ($D_{k1,k2}$), $D_{k1,k2} \in D$ gives the size of a dataset $D_{k1,k2}$, measured in some predetermined unit like bits or bytes.

A big data workflow graph is a weighted directed acyclic graph that includes a set of vertices a.k.a. tasks and a set of edges a.k.a. data dependencies, which represent the output datasets that originate from one task and passed as an input to another task in the workflow. We divide the workflow graph into several partitions a.k.a. levels. Each partition in the workflow graph has a set of vertices and a set of outgoing edges that represent the data passed as input to the next partition.

The weight of the vertices is calculated by the average task computation cost function and the weight of the edges is calculated by the average data communication cost function. More formally, a big data workflow graph is defined as:

Definition 6.3: Given a workflow W in a cloud environment C, a big data workflow graph G, represents a weighted directed acyclic graph with 14-tuple G(T, D, R, F_c, $F_{\bar{c}}$, F_p, $F_{\bar{p}}$, F_m, $F_{\bar{m}}$, F_n, $F_{\bar{n}}$, P, TP, RT), where

- the vertices of the graph represent a set of tasks T.
- the edges of the graph represent a set of data dependencies D.
- R is a set of resources in the cloud environment.
- F_c: $D \times R \times R \rightarrow Q^+0$ is the data communication cost function. F_c ($D_{k1,k2}$, R_{i1}, R_{i2}), $D_{k1,k2}$ $\in$ D, R_{i1}, R_{i2} $\in$ R gives the data communication cost of $D_{k1,k2}$ from resource R_{i1} to resource R_{i2}.
- $F_{\bar{c}}$: $D \rightarrow Q^+0$ is the average data communication cost function. $F_{\bar{c}}$ ($D_{k1,k2}$), $D_{k1,k2}$ $\in$ D gives the average data communication cost of $D_{k1,k2}$ for all the resources R, which is taken as the weight of edge in the graph G. The weight of the edge is 0 for the same resource.
- F_p: $T \times R \rightarrow Q^+$ is the task computation cost function. F_p(T_k, R_i), T_k $\in$ T, R_i $\in$ R gives the computation cost of T_k on resource R_i.
- $F_{\bar{p}}$: $T \rightarrow Q^+$ is the average task computation cost function, $F_{\bar{p}}$(T_k) gives the average computation cost of task T_k, which is taken as the weight of vertex in the graph G.
- F_m: $D \times R \times R \rightarrow Q^+0$ is the data communication time function. F_m($D_{k1,k2}$, R_{i1}, R_{i2}), $D_{k1,k2}$ $\in$ D; R_{i1}, R_{i2} $\in$ R gives the data communication time of $D_{k1,k2}$ from resource R_{i1} to resource R_{i2}.
- $F_{\bar{m}}$: $D \rightarrow Q^+0$ is the average data communication time function. F_m: (k1, k2), $D_{k1,k2}$ $\in$ D gives the average data communication time of $D_{k1,k2}$ for all the resources R.
- F_n: $T \times R \rightarrow Q^+$ is the task computation time function. F_n(T_k, R_i), T_k $\in$ T, R_i $\in$ R gives the computation time of T_k on resource R_i.

- $F_{\bar{n}}$: $T \rightarrow Q^+$ is the average task computation time function, $F_{\bar{n}}(T_k)$ gives the average computation time of task T_k.
- P: $N \rightarrow T$ is the partition task function, P[j] or P_j gives all the tasks of partition j. R_{P_j} represents the set of resources assigned to the tasks in partition P_j.
- TP: $T \rightarrow N$ is the task partition function, TP[T_k] or TP_{T_k} gives the partition number of task T_k.
- RT: $P \rightarrow RT$ is the partition resource type function. RT[P_j] gives the resource type that is assigned to partition P_j.

While executing a workflow on a set of resources in the cloud, there is a cost and time incurred during the execution process. Makespan is the total time taken to complete the workflow execution. Each resource in the cloud has a type associated with it. In addition, there is also a cost associated with each type of resource per unit time. The execution of the workflow on a cheapest resource take more time than executing the workflow on an expensive resource. Because there is a tradeoff between performance of the workflow execution and the cost associated with the workflow execution, we compute the cost associated with the workflow execution at different level of granularity such as the minimum, the average and the maximum completion cost. In order to achieve our objective of minimizing the makespan, we calculate the time taken to complete the execution of each partition in the workflow. More formally, we define the workflow execution environment as:

Definition 6.4: Given a workflow W in a cloud environment C, a workflow execution environment, represents the cost and time incurred during the execution of the workflow with 5-tuple WE (CC, $\bar{CC}$, CT, minCC, maxCC), where

- CC: Partition $\times R \rightarrow Q^+0$ is the workflow partition completion cost function. CC(P_j , R_i), $P_j \in$ Partition gives the sum of task computation cost of all the tasks $T_k \in P_j$ assigned to R_i as well as the data communication cost for all the outgoing edges from all the tasks $T_k \in P_j$. We formally define as:

$$CC(P_j, R_{i1}) = \sum_{k=1}^{K} F_p(T_k, R_{i1}) + \sum_{\substack{i1,i2=1 \\ i1 \neq i2}}^{I} \sum_{\substack{k=1,k1=1 \\ k \neq k1}}^{K} F_c((k, k1), R_{i1}, R_{i2})$$

- $\bar{CC}$: Partition $\rightarrow Q^{+}0$ is the average workflow partition completion cost function. $\bar{CC}(\mathrm{P_j})$, $\mathrm{P_j} \in$ Partition gives the sum of average task computation cost of all the tasks $\mathrm{T_k} \in \mathrm{P_j}$ and the average data communication for all the outgoing edges from all the tasks $\mathrm{T_k} \in \mathrm{P_j}$. We formally define as:

$$\bar{CC}(P_{\mathrm{j}}) = \sum_{k=1}^{K} F_{\bar{p}}(T_{\mathrm{k}}) + \sum_{\substack{k=1,k1=1 \\ k \neq k1}}^{K} F_{\bar{c}}((k,k1))$$

- CT: Partition $\rightarrow Q^{+}0$ is the workflow partition completion time function. $\mathrm{CT}(\mathrm{P_j})$, $\mathrm{P_j} \in$ Partition gives the maximum of average task computation time of all the tasks $\mathrm{T_k} \in \mathrm{P_j}$ and the maximum of average data communication time of all the outgoing edges from all the tasks $\mathrm{T_k} \in \mathrm{P_j}$. We formally define as:

$$CT(P_{\mathrm{j}}) = \max_{T_{\mathrm{k}} \in P_{\mathrm{j}}} F_{\bar{n}}(T_{\mathrm{k}}) + \max_{T_{\mathrm{k}} \in P_{\mathrm{j}}} F_{\bar{m}}(D_{\mathrm{k,k1}})$$

- minCC: Partition $\rightarrow Q^{+}0$ is the minimum workflow partition completion cost function. $\mathrm{minCC}(\mathrm{P_j})$, $\mathrm{P_j} \in$ Partition gives the sum of minimum task computation cost of all the tasks $\mathrm{T_k} \in \mathrm{P_j}$ and the minimum data communication for all the outgoing edges from all the tasks $\mathrm{T_k} \in \mathrm{P_j}$. We formally define minCC as:

$$minCC(P_{\mathrm{j}}) = \sum_{T_{\mathrm{k}} \in P_{\mathrm{j}}} CC(T_{\mathrm{k}}, R_{\mathrm{cheapest}})$$

- maxCC: Partition $\rightarrow Q^{+}0$ is the maximum workflow partition completion cost function. $\mathrm{maxCC}(\mathrm{P_j})$, $\mathrm{P_j} \in$ Partition gives the sum of maximum task computation cost of all the tasks $\mathrm{T_k} \in \mathrm{P_j}$ and the maximum data communication for all the outgoing edges from all the tasks $\mathrm{T_k} \in \mathrm{P_j}$. We formally define maxCC as:

$$maxCC(P_{\mathrm{j}}) = \sum_{T_{\mathrm{k}} \in P_{\mathrm{j}}} CC(T_{\mathrm{k}}, R_{\mathrm{expensive}})$$

At workflow run time, a user driven budget is allocated as a dollar amount for the entire workflow. We compute the sub-budget as a dollar amount for each partition in the workflow based

on the user provided budget. The sub-budget is computed based on the average completion cost for each partition. The partition with the high computation intensive tasks and high data intensive outgoing edges has more sub-budget than the partition with the low computation intensive tasks and the low data intensive outgoing edges. In addition to the sub-budget, there is also a threshold provided to each partition as a dollar amount. The threshold for a partition is computed based on the sub-budget allocated to the next subsequent partition and the completion cost incurred for executing the partition by using the most expensive resource provided by the maximum completion cost. The threshold is set to be always greater than or equal to zero. The partition resource type (PRT) function identifies the most expensive resource type for each partition in a workflow while still meeting the budget constraint. The goal of PRT function is to minimize the completion time of the tasks in each partition. We minimize the workflow makespan by applying PRT for each partition in the workflow.

Definition 6.5: Given a workflow W in a cloud computing environment C and a budget B, a workflow partition cost represents the budget allocated to each partition of the workflow and the actual cost incurred at each partition of the workflow with 6-tuple PC(SB, Threshold, PRT, ACC, Credit, Debit), where

- SB: $Partition \rightarrow Q^{+}$ is the sub-budget function. SB(P_j), $P_j \in$ Partition gives the sub-budget assigned to the partition P_j and can be calculated formally as follows:

$$SB(P_j) = \frac{\bar{CC}(P_j)}{\sum_{j1=1}^{J} \bar{CC}(P_{j1})} \times B$$

- Threshold: $Partition \rightarrow Q^{+}$ is the threshold function. Threshold(P_j), $P_j \in$ Partition gives the threshold assigned to the partition P_j. It can be calculated as follows:

$$Threshold(P_j) = Max\{0, SB(P_{j+1})\} - maxCC(P_{j+1})$$

- PRT: $Partition \times R \times SB \times Threshold \rightarrow RT$ is the partition resource type function that is used to identify the most expensive resource type for each partition. PRT is based on the

criteria that the total completion cost for all the tasks in the partition is less than equal to the sum of the sub-budget and the threshold assigned to the partition.

- ACC: $Partition \rightarrow Q^{+}$ is the actual completion cost that is used to compute the total cost for completing all the tasks in a partition, that are assigned to the resources of a particular resource type. Supposedly, all the tasks in partition j are assigned to RT[j] then ACC can be formally calculated as:

$$ACC[P_{\mathrm{j}}] = \sum CC(P_{\mathrm{j}}, F_{\mathrm{S}}(R_{\mathrm{T}}[P_{\mathrm{j}}]))$$

- Credit: $Partition \rightarrow Q^{+}$ is the credit function. Credit[$\mathrm{P_j}$], $\mathrm{P_j}$ âĹĹ Partition gives the credit assigned to the partition $\mathrm{P_j}$. It can be calculated as follows:

$$Credit[P_{\mathrm{j}}] = Max\{0, SB(P_{\mathrm{j}}) - ACC[P_{\mathrm{j}}]\}$$

- Debit: $Partition \rightarrow Q^{+}$ is the debit function. Debit[$\mathrm{P_j}$], $\mathrm{P_j}$ âĹĹ Partition gives the debit assigned to the partition $\mathrm{P_j}$. It can be calculated as follows:

$$Debit[P_{\mathrm{j}}] = Max\{0, ACC[P_{\mathrm{j}}] - SB(P_{\mathrm{j}})\}$$

Our objective is to minimize workflow makespan while still satisfying the budget constraint. We formally define our objective function and constraints as follows:

Definition 6.6: Given a workflow W in a cloud environment C, and budget B, a workflow makespan minimization represents the objective function to minimize makespan under the given budget constraint.

$$W_{\mathrm{M}} = \sum_{j=1}^{J} \sum_{i=1}^{I} CT(P_{\mathrm{j}}, R_{\mathrm{i}}) \times X_{\mathrm{ji}}$$

where,

$$X_{\mathrm{ji}} = \begin{cases} 1, & \text{if partition } \mathrm{P_j} \text{ is assigned to resource } \mathrm{R_i} \\ 0, & \text{otherwise} \end{cases}$$

such that the following constraints are satisfied:

1. $\sum_{j=1}^{J}\sum_{i=1}^{I} CC(P_{\mathrm{j}}, R_{\mathrm{i}}) \times X_{\mathrm{ji}} \leq B$

2. $\sum_{j=1}^{J}\sum_{i=1}^{I} X_{\mathrm{ji}} = 1$ for all the tasks in partition j assigned to a resource $\mathrm{R_i} \in \mathrm{R}$.

There can be cases as follows:

1. if $\mathrm{B} < \sum_{j=1}^{J} minCC(P_{\mathrm{j}})$, then we cannot satisfy the budget constraint and hence we assign all the partitions to the cheapest resource.

2. if $\mathrm{B} > \sum_{j=1}^{J} maxCC(P_{\mathrm{j}})$, then we can satisfy the budget constraint and hence we assign all the partitions to the most expensive resource.

3. if $\sum_{j=1}^{J} minCC(P_{\mathrm{j}}) \leq B \leq \sum_{j=1}^{J} maxCC(P_{\mathrm{j}})$ then we use our strategy to find the optimal solution.

6.3 The BARENTS Scheduler

Our BARENTS scheduler parses the specification of the workflow and generates a weighted directed acyclic graph with each node representing the tasks and each edge representing the data dependency between the tasks in the workflow. We estimate the number of instructions that exists in each task and the size of the data dependencies that exists between each task for all the workflows in our repository. In addition, we also estimate the number of instructions that can be executed per unit time and the size of the data that can be transferred per unit time for different cloud resource types. The estimates are adjusted automatically during every workflow run to achieve accuracy. The weight of the nodes and the edges in the graph are calculated by using the average computation cost and average data movement cost, respectively based on our estimation. We partition the workflow into different partitions in a topological manner. We validate that the tasks in each partition do not have any data dependency between them and at the same time there is at least one or more data dependency between the tasks in different partitions. After creating the partitions, we assign an initial sub-budget to each partition based on the user defined budget dollar amount provided for the workflow. Our Cloud Resource Manager (CRM) [63, 64] is used to manage the cloud resources by maintaining a catalogue that provides a list of resource types and their associated cost incurred for an hourly rate. We compute the minimum completion cost for each partition, which is the cost

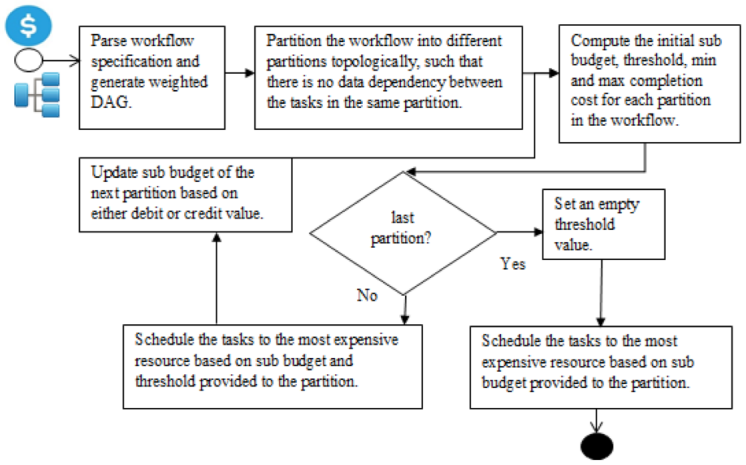


Figure 6.1: The BARENTS flowchart.

incurred for executing all the tasks in a partition. This cost includes both the computation and movement of all the data dependencies using the cheapest resource provided in the catalogue. We compute the maximum completion cost for each partition, which is the cost incurred for executing all the tasks in a partition. This cost includes both the computation and movement of all the data dependencies using the most expensive resource provided in the catalogue.

Resource Type	Instructions per min	Data Movement (MB/Min)	Cost per hour
t2.nano	500	17	0.0064
t2.micro	1000	20	0.013
t2.small	1500	25	0.026
t2.meduim	2000	34	0.052
t2.large	2500	50	0.104

(a)

Task	# of ins (Millions)	$F_{\bar{p}}$ ($)
T_1	1	0.3698
T_2	2	0.7395
T_3	3	1.1093
T_4	4	1.4791
T_5	5	1.8488
T_6	6	2.2187
T_7	7	2.5885

(b)

Partition	Sub budget	minCC	maxCC	Threshold
P_1	0.3825	0.2263	0.7453	2.7878
P_2	7.1323	4.3445	13.9273	0.9686
P_3	2.4852	1.5167	4.8533	0

(d)

Data Dependency	Data Size (MB)	$F_{\bar{e}}$ ($)
$D_{1,2}$	100	0.0019
$D_{1,3}$	200	0.0038
$D_{1,4}$	300	0.0057
$D_{1,5}$	400	0.0076
$D_{1,6}$	500	0.0095
$D_{2,7}$	150	0.0027
$D_{3,7}$	250	0.0047
$D_{4,7}$	350	0.0104
$D_{5,7}$	450	0.0085
$D_{6,7}$	550	0.0104

(c)

Partition	ACC	Credit	Debit	Sub budget	ACT
P_1	0.7453	0	0.3628	0.3825	410
P_2	5.8080	0.9615	0.0	6.7694	4022
P_3	2.0224	0.8257	0.0	2.8481	4667

(e)

Table 6.5: Workflow budget allocation summary.

In addition, we compute a threshold value for each partition that is used as a triggering factor by exploiting the dependency between the partitions. The threshold value for each partition provides more processing power for executing the tasks in each partition by borrowing some budget from the next subsequent partition. By doing so, we are able to schedule the tasks in each partition

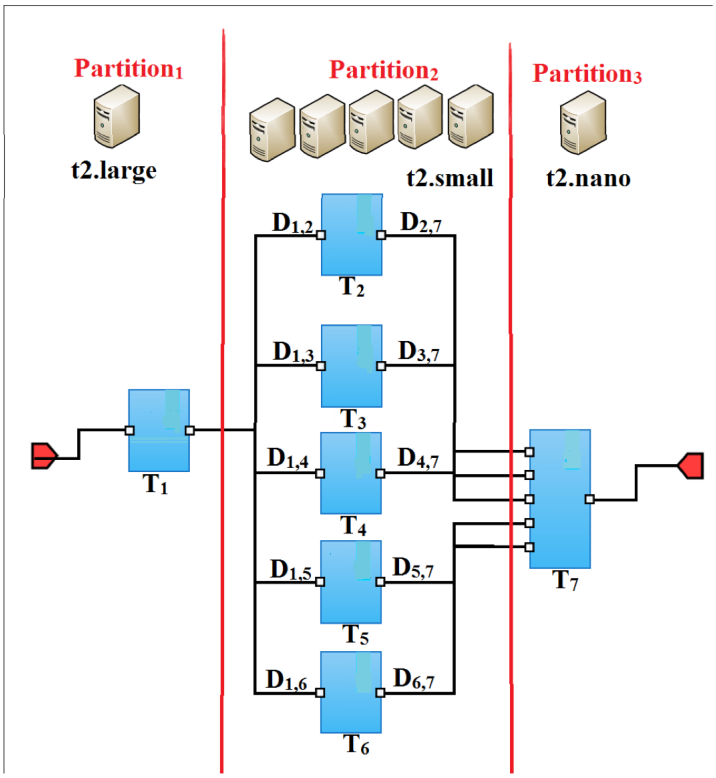


Figure 6.2: An example Workflow w.

to the most expensive resource listed in the catalogue within the range of the combined sub-budget and threshold value assigned to the partition and thereby minimizing the makespan of the partition. After identifying the appropriate resource type, each task in the partition is scheduled to execute in a different set of resources that are of the same resource type. The actual completion cost for each partition is calculated and we compute the credit or debit value based on the actual completion cost and the initial sub-budget provided to the partition. We adjust the sub-budget of the next partition by using the credit or debit value that was calculated for the current partition. The recalculation and adjustment of the sub-budget is done for every partition in the workflow except for the last partition. Because the last partition does not have any subsequent partition to borrow the budget from, the threshold value, the credit value and the debit value is set to be 0 for the last partition. We present the flowchart of our BARENTS scheduler in Figure 6.1.

In Table 6.5(a), we show the resource catalogue maintained by the CRM. For example, we show the 5 resource types with the corresponding number of instructions (in million lines of code) executed per minute, the data movement size (in mega bytes) per minute and the cost in dollar

amount associated with the provisioning of the resource per hour. In Figure 6.2, we show an example workflow that consists of seven tasks and ten data dependencies. First, the BARENTS scheduler parses the specification of the workflow and generates the weighted DAG shown in Table 6.5(b, c). Second, the workflow is partitioned in a topological manner with: P_1={T_1, $D_{1,2}$, $D_{1,3}$, $D_{1,4}$, $D_{1,5}$, $D_{1,6}$}, P_2={T_2-T_6, $D_{2,7}$, $D_{3,7}$, $D_{4,7}$, $D_{5,7}$, $D_{6,7}$} and P_3={T_7}. Next, as presented in Table 6.5(d), we calculate the initial sub-budget, the minimum completion cost, the maximum completion cost and the threshold value for each partition of the workflow. We compute the resource type for partition 1 by finding the most expensive resource, that is "t2.large", with a debit of \$0.3628 and credit of \$0.0. We update the sub-budget of partition 2. We compute the resource type for partition 2 by finding the most expensive resource, that is "t2.small", with a debit of 0.0\$ and credit of \$0.9615. We update the sub-budget of partition 3. We compute the resource type for partition 3 by finding the most expensive resource, which is "t2.nano", with a debit of \$0.0 and credit of \$0.8257. Thereby, we minimized the workflow makespan to 9099 minutes.

We present the pseudo code of our BARENTS scheduler in Algorithm 1. The inputs of the algorithm are the specification of the workflow and a user defined budget dollar amount. The output of the algorithm is the map that consists of all the tasks and the corresponding resources where the tasks are scheduled to execute. In line 4, we parse the specification of the input workflow w and generate a weighted DAG. In line 5, we partition the workflow topologically and generate different partitions. In line 6, we calculate the total completion cost by adding the average completion cost of all the tasks in the workflow w. In lines 7-27, we loop through each partition in the workflow w to identify the appropriate resource for each task in the partition. In line 8, we calculate the sub-budget for the current partition. In lines 9-11, we calculate the sub-budget for the second partition. In lines 12-21, we validate if the current partition is not the last one in the workflow. In line 13, we calculate the maximum completion cost for the next subsequent partition. In line 14, we calculate the threshold for the current partition. In line 15, we calculate the most expensive resource type by calling the partition resource type (PRT) function. In line 16, we assign all the tasks in the partition to different resources of the resource type computed in line 15. The schedule is then added to the output schedule map. In line 17, we compute the actual completion cost for executing all the tasks using the resources assigned to them. In line 18, we calculate the debit value and in line

1: **Algorithm 1** BARENTS Scheduler
2: **input**: workflow *w,* budget *B*
3: **output**: *d*, a map storing task-VM assignments.
4: parse w and generate a weighted DAG (w).
5: *tasksByPartition* ← partition workflow topologically.
6: TCC = $\sum_{j=1}^{J} \sum_{T_k \in P_j} \overline{CC}(T_k, R)$
7: **for each** partition $P_j \in$ *tasksByPartition*
8: *SB* [P_j] = $\sum_{T_k \in P_j} \overline{CC}(T_k, R)$ / TCC * B
9: **if** (P_j is first partition) **then**
10: *SB* [P_{j+1}] = $\sum_{T_k \in P_{j+1}} \overline{CC}(T_k, R)$ / TCC * B
11: **end if**
12: **if** (P_j is not last partition) **then**
13: *maxCC* [P_{j+1}] = $\sum_{T_k \in P_{j+1}} CC(T_k, R_{expensive})$
14: *Thres* [P_j] = Max {0, *SB* [P_{j+1}] – *maxCC* [P_{j+1}]}
15: RT [P_j] = PRT (P_j, R, *SB* [P_j] + *Thres* [P_j])
16: d ← d ∪ MAP (T_k, F_S(RT [P_j])) ∀ $T_k \in P_j$
17: ACC [P_j] = $CC(P_j, F_S(RT[P_j]))$
18: DEBIT [P_j] = Max{0, ACC [P_j] – SB [P_j]}
19: SB [P_{j+1}] = SB [P_{j+1}] – DEBIT [P_j]
20: CREDIT [P_j] = Max{0, SB[P_j] – ACC [P_j]}
21: SB [P_{j+1}] = SB [P_{j+1}] + CREDIT [P_j]
22: **else if** (P_j is last partition) **then**
23: SB [P_j] = B - $\sum_{j1=1}^{J-1}$ ACC [P_{j1}]
24: RT [P_j] = PRT (P, R, SB [P_j])
25: d ← d ∪ MAP (T_k, F_S(RT [P_j])) ∀ $T_k \in P_j$
26: **end if**
27: **end for**
28: **return d**
29: **end function**

Figure 6.3: The BARENTS Scheduler Algorithm.

20 we calculate the credit value. In lines 19 and 21, we recalculate the sub-budget and make the necessary adjustment based on the credit or debit value. In lines 22-25, we validate whether the current partition is the last partition of the workflow. In line 23, we calculate the sub-budget of the last partition by subtracting the sum of the actual costs of all the previous partitions from the given budget B. In line 24, we calculate the most expensive resource type by calling the partition resource type (PRT) function. In line 25, we assign all the tasks in the last partition to different resources of the resource type computed in line 24. The schedule is then added to the output schedule map. Finally, in line 28, we output the final schedule that consists of all the tasks and the corresponding resources where the tasks are scheduled to execute and exit the code.

6.4 Experimental Discussion

In our DATAVIEW system, we implemented a big data workflow that is from automotive domain. We evaluated the BARENTS scheduler using the OpenXC Autoanalytics workflow [59]. OpenXC is an open source platform that produces 19 signal oriented data trace files from the vehicle automatically at periodic intervals. As mentioned in [63, 64], there is an exponential growth in the data size as the number of miles a vehicle is driven increases with the time for each driver. On average, every year the growth of the data exceeds 14 Eb. OpenXC data analysis is extremely useful for the automotive insurance companies to analyze the driving behavior of their customers by analyzing the large datasets generated from their registered vehicles. Since the analytics is performed using the cloud resources, there is a cost associated based on the computation and data intensity of the tasks in the analytics workflow. There is often a need to minimize the execution time that is taken to perform the analytics based on the budget provided for the analytics by the user. The BARENTS scheduler automatically learns the complexity of the tasks computation and the data transfer between the tasks from an initial estimate. During every workflow run, the accuracy of the complexity level estimates is improved by automatic adjustment of the actual execution measurements.

We performed our experiments in the Amazon EC2 cloud computing environment, which provides a framework to provision and deprovision virtual machines (instances) that are of heterogeneous instance types. The Amazon EC2 cloud environment offers a total of 39 different instance types that are of varied CPU, memory, storage and networking capacity. Each type of instances

consists of an hourly cost for resource utilization and the execution time is based on the complexity level of the analytics workload. For example, the instance types in the general purpose T2 category are listed as: “t2.nano”, “t2.micro”, “t2.small”, “t2.medium”, “t2.large”. The performance of the analytics using the resources of type “t2.nano” for a given workload is the slowest and the cheapest option in terms of cost. On the other hand, for the same workload the performance using the resources of type “t2.large” is the fastest and the most expensive option in terms of cost.

Workflow	No of drivers	Computation Size (MLOC)	Data Size (GB)	Budget ($)
w_1	5	1	1	45
w_2	10	2	2	60
w_3	15	3	3	70
w_4	20	4	4	80
w_5	25	5	5	85
w_6	30	6	6	95
w_7	35	7	7	110
w_8	40	8	8	115
w_9	45	9	9	130
w_{10}	50	10	10	150

Table 6.6: Workload details for OpenXC workflows.

We used two approaches to evaluate the strength of our BARENTS algorithm. The first one is the Workflow Responsive Resource Provisioning and Scheduling (WRPS) [58] algorithm that assigns sub deadline to each bag of tasks and schedules them on to a heterogeneous type of cloud resources. In WRPS, the authors model the problem as an unbounded knapsack minimization problem. The WRPS algorithm is the most noted recent work in the field of workflow scheduling. One limitation of WRPS is that the workflow schedule is generated under a simulated workflow execution environment. In addition, the workflow tasks are assumed to be homogeneous, whereas in reality tasks in a workflow are heterogeneous [63] with different types of tasks that consist of the component code such as the command line application, the web service based application, etc. Hence, WRPS does not consider any optimization strategies for heterogeneous tasks in a workflow. In contrast, using our approach, we model the execution time and execution cost of each heterogeneous task in a workflow by considering both the complexity of the computation in the task, as well as the outgoing data dependencies for each task which are combined for each partition in the

workflow. However, we are still interested in comparing BARENTS to WRPS, when both are using homogeneous workflow tasks in each partition as a bag of tasks. In order to make our comparison realistic, we implemented a slight variation to the original WRPS algorithm by implementing the algorithm with an objective to minimize the makespan under a user driven budget constraint.

The second approach is a slight variation of our BARENTS algorithm called BARENTS*, in which we relaxed the dependencies between the partitions by setting the threshold, the credit and the debit to be 0. The WRPS algorithm provides an optimization to the bag of tasks by scheduling the tasks in a bag to different types of machines. In contrast to WRPS, BARENTS assigns all the tasks inside a given partition to the same type of machine. By performing this comparison, we are able to validate how the partition dependencies and run time sub-budget adjustment proposed in BARENTS is used as a distinguishing feature to outperform WRPS.

BARENTS was evaluated using 10 distinctive workflows developed in the OpenXC domain with different levels of complexity and with different dollar amounts provided as budget. In Table 6.6, we presented all the 10 workflows with their complexity levels such as the computation and data intensity of all the tasks in each of the workflow and the user defined budget. We did the experiments by varying the types of machines (the K value) and presented the measurements from both the cost and makespan perspectives. In Figure 6.4, we show that BARENTS outperforms WRPS by roughly 6-10% margin as the complexity of the workflow increases from w3 to w10. For workflows between w1 and w3, which is of least complexity, WRPS outperforms BARENTS. The reason for this behavior is that WRPS schedules tasks in each bag to the resources of different machine types. The local optimization done at each partition outperforms the global optimization performed by BARENTS when the complexity level is low. We evaluated the behavior of all three approaches and have demonstrated the makespan minimization by varying the number of instance types for K = 5, 10, 15, 20, 25.

In order to vary the K values for same set of workflows with the same set of budgets, we created a bigger range with a larger difference between the instance types and then added new instance types within that range. In Figure 6.4, we illustrate the resource utilization in the cloud for different settings of K. The BARENTS algorithm outperforms WRPS because resources are utilized to the maximum extent for the tasks in each partition. Optimal resource utilization is

achieved because BARENTS sets up partition dependency based on a system driven threshold and automatically adjusts the sub-budget at run time with a system driven credit or debit value, that is calculated from the actual completion cost of the previous partition. As K increases, there is a consistent improvement in makespan minimization and resource utilization.

6.5 Chapter Summary

In this chapter, we proposed a new Big dAta woRkflow schEduler uNder budgeT conStraint known as BARENTS that supports high-performance workflow scheduling in a heterogeneous cloud computing environment with a single objective to minimize the workflow makespan under a provided budget constraint. Our case study and experiments show the competitive advantages of our proposed scheduler. The proposed BARENTS scheduler is implemented in a new release of DATAVIEW, one of the most usable big data workflow systems in the community.

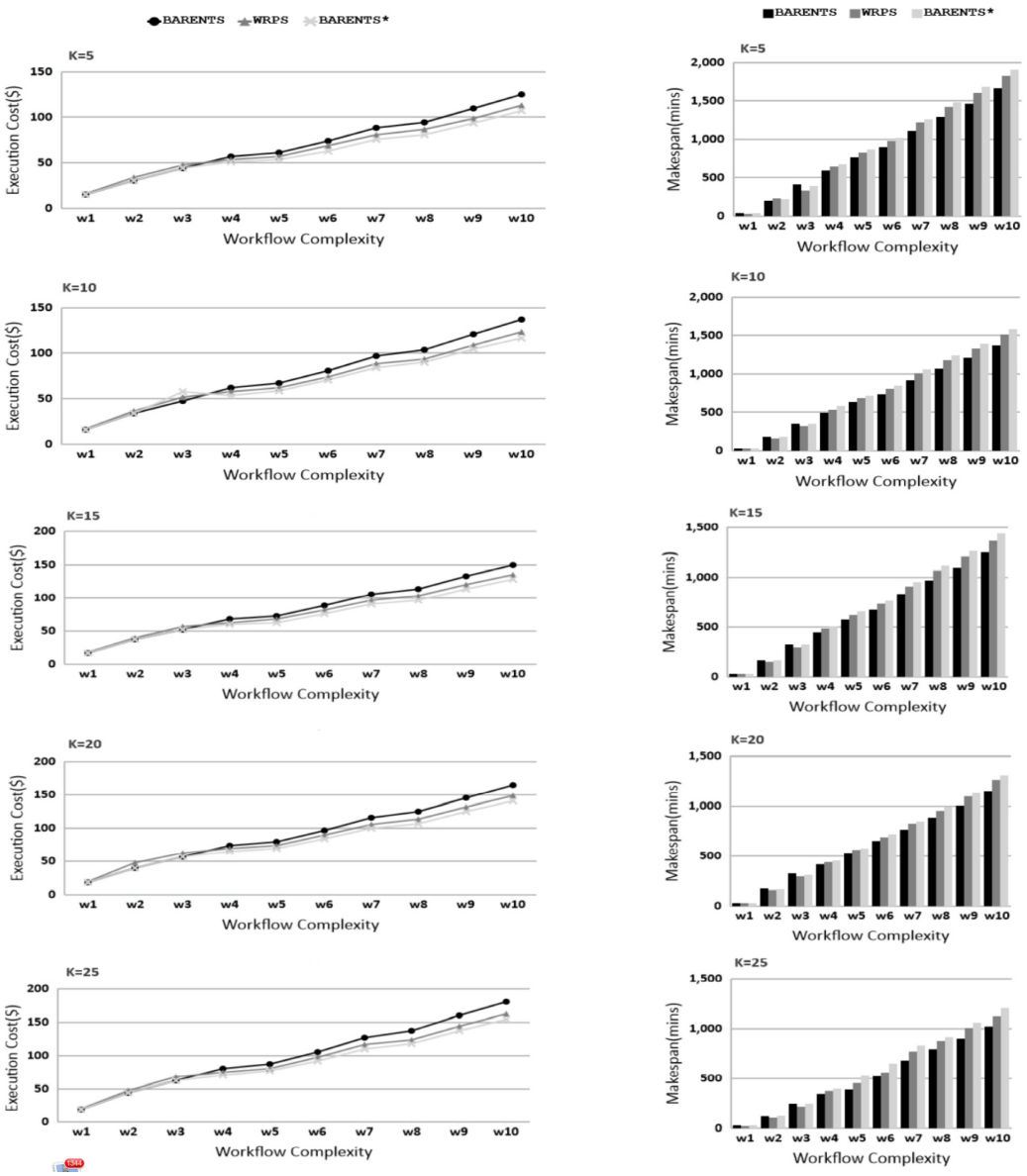


Figure 6.4: Resource utilization and makespan minimization.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

We proposed a new primitive workflow model, to overcome the limitations of the previous multiple-component based task model, in which we separate the registration from the configuration of p-workflow, thereby making the registration process simple and usable. Our new model eases the registration process and hence there is no need to do mapping between multiple task components inside the p-workflow. Second, we proposed a shim generation algorithm to solve the shimming problem raised in Web services by automatically inserting invisible shims and wrapping it around the p-workflow. Third, we integrated MongoDB , an open source document oriented database, into our DATAVIEW system in order to support big data management and processing. Finally, we implemented the proposed models and algorithm in our DATAVIEW system and presented a case study to validate them. Ongoing work includes extension of the primitive workflow model to support registration and execution of a p-workflow in cloud and grid computing environments.

We first developed a workflow recommendation framework to recommend the list of syntactically and semantically compatible workflow candidates and thereby improve the scientific workflow design productivity of the user. Second, we presented a folksonomy based workflow model to extend our previously proposed primitive workflow model that emphasizes on the semantic information in a workflow. Third, we proposed two workflow recommendation algorithms, to capture the social annotations capability on syntactic and semantic workflow recommendation respectively. Finally, we implemented the proposed environment and strategies in our DATAVIEW system and validated our approach with a case study and experimental results. Ongoing work includes extending our recommendation framework to support a proactive, system driven recommendation approach to provide recommendations for all the incomplete workflows in an in-progress workflow.

We proposed a NoSQL data model that: 1) supports high-performance MapReduce-style workflows that automate data partitioning and data-parallelism execution. In contrast to the traditional MapReduce framework, our MapReduce-style workflows are fully composable with other workflows enabling dataflow applications with a richer structure; 2) automates virtual machine provisioning and deprovisioning on demand according to the sizes of input datasets; 3) enables a flexible framework for workflow executors that take advantage of the proposed NoSQL data model to improve the performance of workflow execution. We presented a case study and experiments that show

the competitive advantages of our proposed NoSQL collectional data model and the cloud workflow executors. Ongoing work includes implementing a new set of workflow constructs that can be used for efficient parallel processing.

To schedule big data workflows in the cloud computing environment, we formalize a model of the cloud computing environment and a workflow graph model for the environment. Based on the models, we propose a new Big dAta woRkflow schEduler uNder budgeT conStraint known as BARENTS that supports high-performance workflow scheduling in a heterogeneous cloud computing environment with a single objective to minimize the workflow makespan under a provided budget constraint. Our case study and experiments not only show the competitive advantages of our proposed scheduler, but also enables resources to scale elastically during workflow execution. The proposed BARENTS scheduler is implemented in a new release of DATAVIEW, one of the most usable big data workflow systems in the community

APPENDIX A: SCIENTIFIC WORKFLOW LANGUAGE (SWL 3.0)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http:
www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
<xsd:element name="workflowSpec">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="workflow" type="WorkflowXMLElementType" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:complexType name="WorkflowXMLElementType">
<xsd:sequence>
<xsd:element name="workflowInterface">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="workflowDescription" type="xsd:string" minOccurs="0"/>
<xsd:element name="inputPorts">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="inputPort" type="PortXMLElementType" minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="inputParameter" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:extension>
</xsd:simpleContent>
```

```
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="outputPorts">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="outputPort" type="PortXMLElementType" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
<xsd:attribute name="number" type="xsd:int"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="workflowBody">
<xsd:complexType>
<xsd:choice>
<xsd:sequence>
<xsd:element name="baseWorkflow" type="xsd:string"/>
<xsd:element name="unary-construct">
<xsd:complexType>
<xsd:sequence>
<xsd:choice minOccurs="0" maxOccurs="unbounded">
<xsd:element name="map" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:attribute name="mapPort" type="xsd:string"/>
</xsd:complexType>
```

```
</xsd:element>
<xsd:element name="reduce" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:attribute name="basePort" type="xsd:string"/>
<xsd:attribute name="reducePort" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="tree" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:attribute name="leftPort" type="xsd:string"/>
<xsd:attribute name="rightPort" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="loop" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:attribute name="loopPort" type="xsd:string"/>
<xsd:attribute name="predicate" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="conditional" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:attribute name="conditionalPort" type="xsd:string"/>
<xsd:attribute name="predicate" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="curry" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="inputMapping" type="WorkflowPortMappingXMLElementType" minOccurs="0"
```

```
maxOccurs="unbounded"/>
<xsd:element name="assign" type="WorkflowPortMappingXMLElementType" minOccurs="0" maxOccurs="unb
<xsd:element name="outputMapping" type="WorkflowPortMappingXMLElementType" minOc-
curs="0" maxOccurs="unbounded"/>
</xsd:sequence>
<xsd:attribute name="curryPort" type="xsd:string"/>
<xsd:attribute name="parameter" type="xsd:string"/>
<xsd:attribute name="parameterType" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:choice>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:sequence>
<xsd:element name="taskComponent">
<xsd:complexType>
<xsd:sequence>
<xsd:choice>
<xsd:sequence>
<xsd:element name="wsdlURI" type="xsd:string"/>
<xsd:element name="serviceName" type="xsd:string"/>
<xsd:element name="operationName" type="xsd:string"/>
</xsd:sequence>
<xsd:sequence>
<xsd:element name="directory" type="xsd:string"/>
<xsd:element name="appName" type="xsd:string"/>
</xsd:sequence>
```

```
<xsd:sequence>
<xsd:element name="executable" type="xsd:string"/>
<xsd:element name="appName" type="xsd:string"/>
</xsd:sequence>
</xsd:choice>
<xsd:element name="taskInvocation">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="operatingSystem">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="Windows"/>
<xsd:enumeration value="Unix"/>
<xsd:enumeration value="Linux"/>
<xsd:enumeration value="Mac"/>
<xsd:enumeration value="Unknown"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="invocationMode">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="Local"/>
<xsd:enumeration value="Remote"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="interactionMode">
<xsd:simpleType>
```

```
<xsd:restriction base="xsd:string">
<xsd:enumeration value="Yes"/>
<xsd:enumeration value="No"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="invocationAuthentication" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="hostName" type="xsd:string"/>
<xsd:element name="userName" type="xsd:string"/>
<xsd:element name="password" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="inputParams" type="xsd:string"/>
<xsd:element name="outputParams" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="taskType" use="required">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="WindowsApplication"/>
<xsd:enumeration value="LinuxApplication"/>
<xsd:enumeration value="CommandLine"/>
<xsd:enumeration value="WebService"/>
<xsd:enumeration value="GridJob"/>
```

```
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:complexType>
</xsd:element>
<xsd:element name="T2W">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="inputs">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="input" type="TaskPortMappingXMLElementType" maxOccurs="unbounded
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="outputs">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="output" type="TaskPortMappingXMLElementType" maxOccurs="unbounde
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
 <xsd:sequence>
<xsd:element name="workflowGraph">
<xsd:complexType>
```

<xsd:sequence>

<xsd:element name="workflowInstances">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="workflowInstance" minOccurs="0" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="workflow" type="xsd:string"/>

</xsd:sequence>

<xsd:attribute name="id" type="xsd:string" use="required"/>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

<xsd:element name="dataChannels">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="dataChannel" type="DataChannelXMLElementType" minOccurs="0" maxOccurs="unbou

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

<xsd:element name="G2W">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="inputMapping" type="WorkflowPortMappingXMLElementType" minOccurs="0"

```
maxOccurs="unbounded"/>
<xsd:element name="outputMapping" type="WorkflowPortMappingXMLElementType" minOc-
curs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:sequence>
<xsd:element name="builtin" type="xsd:string"/>
</xsd:sequence>
</xsd:choice>
<xsd:attribute name="mode" use="required">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="unary-construct-based"/>
<xsd:enumeration value="primitive"/>
 <xsd:enumeration value="graph-based"/>
<xsd:enumeration value="builtin"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string"/>
<xsd:attribute name="root" type="xsd:boolean" use="required"/>
</xsd:complexType>
<xsd:complexType name="PortXMLElementType">
<xsd:sequence>
```

```
<xsd:element name="portID" type="xsd:string"/>
<xsd:element name="portName" type="xsd:string"/>
<xsd:element name="portType">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="String"/>
<xsd:enumeration value="Decimal"/>
<xsd:enumeration value="Integer"/>
<xsd:enumeration value="NonPositiveInteger"/>
<xsd:enumeration value="NegativeInteger"/>
<xsd:enumeration value="NonNegativeInteger"/>
<xsd:enumeration value="UnsignedLong"/>
<xsd:enumeration value="UnsignedInt"/>
<xsd:enumeration value="UnsignedShort"/>
<xsd:enumeration value="UnsignedByte"/>
<xsd:enumeration value="PositiveInteger"/>
<xsd:enumeration value="Double"/>
<xsd:enumeration value="Float"/>
<xsd:enumeration value="Long"/>
<xsd:enumeration value="Int"/>
<xsd:enumeration value="Short"/>
<xsd:enumeration value="Byte"/>
<xsd:enumeration value="Boolean"/>
<xsd:enumeration value="Uri"/>
<xsd:enumeration value="Blob"/>
<xsd:enumeration value="Date"/>
<xsd:enumeration value="List"/>
<xsd:enumeration value="Mongo"/>
<xsd:enumeration value="Uri"/>
```

<xsd:enumeration value="File"/>

<xsd:enumeration value="RelationBase"/>

<xsd:enumeration value="CollectionBase"/>

<xsd:enumeration value="Object"/>

</xsd:restriction>

</xsd:simpleType>

</xsd:element>

<xsd:element name="portParameter" type="xsd:string" minOccurs="0"/>

<xsd:element name="portDescription" type="DescriptionXMLElementType" minOccurs="0"/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="TaskPortMappingXMLElementType">

<xsd:attribute name="id" type="xsd:string" use="required"/>

<xsd:attribute name="mode" type="xsd:string" use="required"/>

<xsd:attribute name="name" type="xsd:string" use="required"/>

<xsd:attribute name="type" type="xsd:string" use="required"/>

</xsd:complexType>

<xsd:complexType name="DataChannelXMLElementType">

<xsd:attribute name="type">

<xsd:simpleType>

<xsd:restriction base="xsd:string">

<xsd:enumeration value="OneToOneDataChannel"/>

<xsd:enumeration value="OneToManyDataChannel"/>

<xsd:enumeration value="ManyToOneDataChannel"/>

</xsd:restriction>

</xsd:simpleType>

</xsd:attribute>

<xsd:attribute name="from" type="xsd:string" use="required"/>

<xsd:attribute name="to" type="xsd:string" use="required"/>

```
</xsd:complexType>
<xsd:complexType name="WorkflowPortMappingXMLElementType">
<xsd:attribute name="from" type="xsd:string" use="required"/>
<xsd:attribute name="to" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:simpleType name="DescriptionXMLElementType">
<xsd:restriction base="xsd:string"/>
</xsd:simpleType>
</xsd:schema>
```

APPENDIX B: DATA PRODUCT LANGUAGE (DPL 3.0)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="1.0.0">
<xsd:element name="dataProduct" type="DataProductXMLElementType"/>
<xsd:complexType name="DataProductXMLElementType">
<xsd:sequence>
<xsd:element name="description" type="xsd:string"/>
<xsd:element name="type">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="ScalarValue"/>
<xsd:enumeration value="XmlElement"/>
<xsd:enumeration value="File"/>
<xsd:enumeration value="ColVal"/>
<xsd:enumeration value="List"/>
<xsd:enumeration value="Mongo"/>
<xsd:enumeration value="Relation"/>
<xsd:enumeration value="Collection"/>
<xsd:enumeration value="Dropbox"/>
 </xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="data" type="DataXMLElementType"/>
</xsd:sequence>
<xsd:attribute name="name"/>
</xsd:complexType>
<xsd:complexType name="DataXMLElementType">
<xsd:choice>
```

```
<xsd:element name="scalarValue" type="ScalarValueXMLElementType"/>
<xsd:element name="xmlElement" type="XmlElementXMLElementType"/>
<xsd:element name="blob" type="xsd:base64Binary"/>
<xsd:element name="file" type="FileXMLElementType"/>
<xsd:element name="list" type="ListXMLElementType"/>
<xsd:element name="mongo" type="MongoXMLElementType"/>
<xsd:element name="collectionvalue" type="ColValXMLElementType"/>
<xsd:element name="relation" type="RelationXMLElementType"/>
<xsd:element name="collection" type="CollectionXMLElementType"/>
<xsd:element name="dropboxItem" type="DropboxXMLElementType"/>
 </xsd:choice>
</xsd:complexType>
<xsd:complexType name="ScalarValueXMLElementType">
<xsd:sequence>
<xsd:element name="scalarType" type="ScalarDataTypeEnumeration"/>
<xsd:element name="value" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="ScalarDataTypeEnumeration">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="String"/>
<xsd:enumeration value="Decimal"/>
<xsd:enumeration value="Integer"/>
<xsd:enumeration value="NonPositiveInteger"/>
<xsd:enumeration value="NegativeInteger"/>
<xsd:enumeration value="NonNegativeInteger"/>
<xsd:enumeration value="UnsignedLong"/>
<xsd:enumeration value="UnsignedInt"/>
<xsd:enumeration value="UnsignedShort"/>
```

```
<xsd:enumeration value="UnsignedByte"/>
<xsd:enumeration value="PositiveInteger"/>
<xsd:enumeration value="Double"/>
<xsd:enumeration value="Float"/>
<xsd:enumeration value="Long"/>
<xsd:enumeration value="Int"/>
<xsd:enumeration value="Short"/>
<xsd:enumeration value="Byte"/>
<xsd:enumeration value="Boolean"/>
<xsd:enumeration value="Uri"/>
<xsd:enumeration value="Blob"/>
<xsd:enumeration value="Date"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="ListXMLElementType">
<xsd:sequence minOccurs="0" maxOccurs="unbounded">
<xsd:element name="listName" type="xsd:string"/>
<xsd:element name="listType" type="xsd:string"/>
<xsd:element name="listValue" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DropboxXMLElementType">
<xsd:sequence minOccurs="0" maxOccurs="unbounded">
<xsd:element name="dropboxAppKey" type="xsd:string"/>
<xsd:element name="dropboxAppSecret" type="xsd:string"/>
<xsd:element name="dropboxAppToken" type="xsd:string"/>
<xsd:element name="dropboxPath" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
```

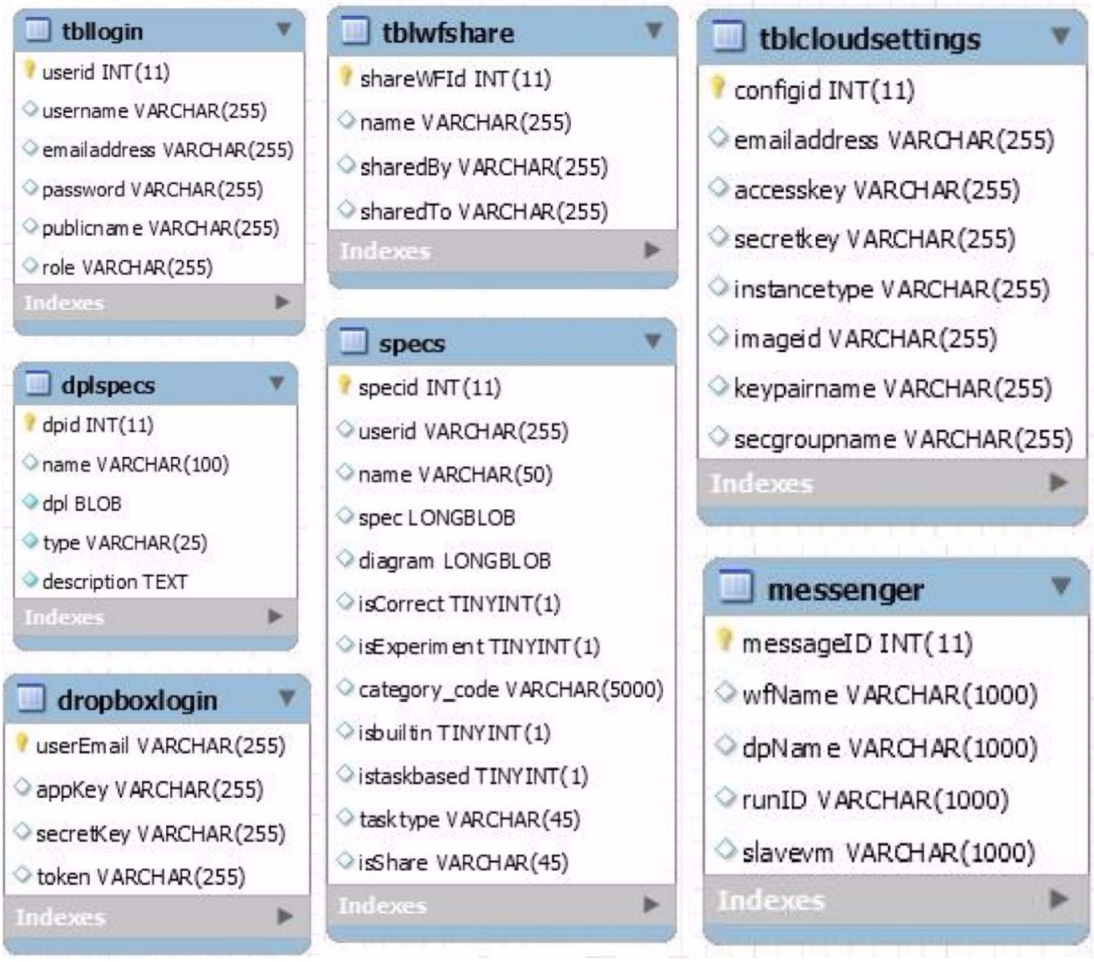
```
<xsd:complexType name="FileXMLElementType">
<xsd:sequence minOccurs="0" maxOccurs="unbounded">
<xsd:element name="FileName" type="xsd:string"/>
<xsd:element name="FilePath" type="xsd:string"/>
<xsd:element name="FileType" type="xsd:string"/>
<xsd:element name="FileStorage" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="MongoXMLElementType">
<xsd:sequence minOccurs="0" maxOccurs="unbounded">
<xsd:element name="colName" type="xsd:string"/>
<xsd:element name="dbName" type="xsd:string"/>
<xsd:element name="hostName" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ColValXMLElementType">
<xsd:sequence minOccurs="0" maxOccurs="unbounded">
<xsd:element name="ValueType" type="xsd:string"/>
<xsd:element name="Value" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="XmlElementXMLElementType">
<xsd:sequence>
<xsd:element name="value" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="RelationXMLElementType">
<xsd:sequence>
<xsd:element name="schema">
```

```
<xsd:complexType>
<xsd:sequence maxOccurs="unbounded">
<xsd:element name="column" type="DataColumnXMLElementType"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:choice>
<xsd:element name="instance">
<xsd:complexType>
<xsd:sequence maxOccurs="unbounded">
<xsd:element name="row" type="DataRowXMLElementType"/>
</xsd:sequence>
<xsd:attribute name="count" type="xsd:integer"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="DBEntry">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="dbName" type="xsd:string"/>
<xsd:element name="tableName" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:choice>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DataColumnXMLElementType">
<xsd:sequence>
<xsd:element name="columnName" type="xsd:string"/>
```

```
<xsd:element name="columnType" type="ScalarDataTypeEnumeration"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DataRowXMLElementType">
<xsd:sequence maxOccurs="unbounded">
<xsd:element name="dataElement" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CollectionXMLElementType">
<xsd:sequence>
<xsd:element name="schema">
<xsd:complexType>
<xsd:sequence maxOccurs="unbounded">
<xsd:element name="key" maxOccurs="unbounded">
<xsd:complexType>
<xsd:attribute name="name" type="xsd:string"/>
<xsd:attribute name="type" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="value">
<xsd:complexType>
<xsd:attribute name="name" type="xsd:string"/>
<xsd:attribute name="type" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="DBEntry">
```

```
<xsd:complexType>
<xsd:sequence maxOccurs="unbounded">
<xsd:element name="Entry" type="DBEntryXMLElementType"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DBEntryXMLElementType">
<xsd:sequence minOccurs="0" maxOccurs="unbounded">
<xsd:element name="dbName" type="xsd:string"/>
<xsd:element name="tableName" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="PairXMLElementType">
<xsd:sequence>
<xsd:element name="key" type="xsd:string"/>
<xsd:choice>
<xsd:element name="relation" type="RelationXMLElementType"/>
<xsd:element name="collection" type="CollectionXMLElementType"/>
 <xsd:element name="scalarValue" type="ScalarValueXMLElementType"/>
</xsd:choice>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

APPENDIX C: ENTITY RELATIONSHIP DIAGRAM (ERD 3.0)



Tables Description:

1- tbllogin: This table stores the information of the users
2- dplspecs: This table stores the information of data products
3- dropboxlogin: This table stores the information of user's Dropbox
4- tblwfshare: This table stores the information of workflow sharing
5- specs: This table stores the information of workflow specification
6- tblclludsettings: This table stores the information of user's cloud
7- messenger: This table stores the information of cloud virtual machines

Figure 7.1: ER Diagram of DATAVIEW.

APPENDIX D: KNN ALGORITHM PRIMITIVE WORKFLOW

Workflow Specification

<workflowSpec>

<workflow name="KNN" root="true">

<workflowInterface>

<workflowDescription>

simple KNN workflow</workflowDescription>

<inputPorts>

<inputPort>

<portID>i1</portID>

<portName>a</portName>

<portType>File</portType>

<portDescription>port i1 description</portDescription>

</inputPort>

<inputPort>

<portID>i2</portID>

<portName>b</portName>

<portType>File</portType>

<portDescription>port i2 description</portDescription>

</inputPort>

<inputPort>

<portID>i3</portID>

<portName>c</portName>

<portType>Integer</portType>

<portDescription>port i3 description</portDescription>

</inputPort>

</inputPorts>

<outputPorts>

<outputPort>

<portID>o1</portID>

<portName>c</portName>

<portType>File</portType>

<portDescription>port o1 description</portDescription>

</outputPort>

</outputPorts>

</workflowInterface>

<workflowBody mode="builtin">

<builtin>KNN</builtin>

</workflowBody>

</workflow>

</workflowSpec>

APPENDIX E: APRIORI ALGORITHM PRIMITIVE WORKFLOW

Workflow Specification

<workflowSpec>

<workflow name="APRIORI" root="true">

<workflowInterface>

<workflowDescription>

simple APRIORI workflow</workflowDescription>

<inputPorts>

<inputPort>

<portID>i1</portID>

<portName>a</portName>

<portType>File</portType>

<portDescription>port i1 description</portDescription>

</inputPort>

<inputPort>

<portID>i2</portID>

<portName>b</portName>

<portType>Integer</portType>

<portDescription>port i2 description</portDescription>

</inputPort>

<inputPort>

<portID>i3</portID>

<portName>c</portName>

<portType>Double</portType>

<portDescription>port i3 description</portDescription>

</inputPort>

</inputPorts>

<outputPorts>

<outputPort>

<portID>o1</portID>

<portName>c</portName>

<portType>File</portType>

<portDescription>port o1 description</portDescription>

</outputPort>

</outputPorts>

</workflowInterface>

<workflowBody mode="builtin">

<builtin>APRIORI</builtin>

</workflowBody>

</workflow>

</workflowSpec>

APPENDIX F: KMEANS ALGORITHM PRIMITIVE WORKFLOW

Workflow Specification

<workflowSpec>

<workflow name="KMEANS" root="true">

<workflowInterface>

<workflowDescription>

simple KMEANS workflow</workflowDescription>

<inputPorts>

<inputPort>

<portID>i1</portID>

<portName>a</portName>

<portType>File</portType>

<portDescription>port i1 description</portDescription>

</inputPort>

<inputPort>

<portID>i2</portID>

<portName>b</portName>

<portType>Integer</portType>

<portDescription>port i2 description</portDescription>

</inputPort>

<inputPort>

<portID>i3</portID>

<portName>c</portName>

<portType>Integer</portType>

<portDescription>port i3 description</portDescription>

</inputPort>

</inputPorts>

<outputPorts>

<outputPort>

<portID>o1</portID>

<portName>c</portName>

<portType>File</portType>

<portDescription>port o1 description</portDescription>

</outputPort>

</outputPorts>

</workflowInterface>

<workflowBody mode="builtin">

<builtin>KMEANS</builtin>

</workflowBody>

</workflow>

</workflowSpec>

REFERENCES

[1] J. Ambite, D. Kapoor. Automatically composing data workflows with relational descriptions and shim services. *In Proc. of the ISWC/ASWC Conference*, pages 15âĂŞ29, 2007.

[2] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. VisTrails: visualization meets data management. *In Proc. of the SIGMOD Conference*, pages 745âĂŞ747, 2006.

[3] S. Davidson, J. Freire. Provenance and scientific workflows: challenges and opportunities. *In Proc. of the SIGMOD Conference*, pages 1345âĂŞ1350, 2008.

[4] E. Deelman, A. Chervenak. Data management challenges of data-intensive scientific workflows. *In Proc. of the CCGRID Conference*, pages 687âĂŞ692, 2008.

[5] D. Hull, R. Stevens, and P. Lord. Describing web services for user-oriented retrieval. *In Proc. of the W3C Workshop on Frameworks for Semantics in Web Services*, pages 9âĂŞ10, 2005.

[6] D. Hull, R. Stevens, P. Lord, C. Wroe, and C. Goble. Treating shimantic web syndrome with ontologies. *In Proc. of the AKT- SWS04 Workshop*, 2004.

[7] C. Lin, S. Lu, X. Fei, A. Chebotko, Z. Lai, D. Pai, F. Fotouhi, and J. Hua. A reference architecture for scientific workflow management systems and the VIEW SOA solution. *IEEE Transactions on Services Computing*, 2(1):79âĂŞ92, 2009.

[8] C. Lin, S. Lu, Z. Lai, A. Chebotko, X. Fei, J. Hua, and F. Fotouhi. Service-oriented architecture for VIEW: A visual scientific workflow management system. *In Proc. of the IEEE SCC Conference*, pages 335âĂŞ342, 2008.

[9] B. Ludascher, S. Bowers, T. McPhillips, and N. Podhorszki. Scientific workflows: More e-science mileage from cyberinfrastructure. *In Proc. of the e-Science and Grid Computing Conference*, pages 145âĂŞ152, 2006.

[10] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045âĂŞ 3054, 2004.

[11] M.Szomszor, T. Payne, and L. Moreau. Automated syntactic medation for web service integration. *In Proc. of the ICWS Conference*, pages 127âĂŞ136, 2006.

[12] C. Lin, S. Lu, X. Fei, D. Pai and J. Hua, A Task Abstraction and Mapping Approach to the Shimming Problem in Scientific Workflows, *In Proc. of the IEEE International Conference on Services Computing (SCC)*, pages 284-291, Bangalore, India, 2009.

[13] A. Kashlev, S. Lu, and A. Chebotko, Coercion Approach to the Shimming Problem in Scientific Workflows, *In Proc. of the IEEE International Conference on Services Computing (SCC)*, pages 416-423, 2014, Santa Clara, CA.

[14] S. Singh, N. Singh, Big Data Analytics, *In Proc. of the IEEE International Conference on Communication, Information and Computing Technology (ICCICT)*, Mumbai, India, 2012.

[15] http://www.01.ibm.com/software/data/infosphere/hadoop/what-is-big-data-analytics.html

[16] S. Xu, S. Bao, B. Fei, Z. Su, and Y. Yu, Exploring folksonomy for personalized search, *In Proc. of the 31st annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2008)*, pages 155-162, 2008.

[17] A. Hotho, R. JÃď schke, C. Schmitz, and G. Stumme, Information retrieval in folksonomies: search and ranking, *In Proc. of the 3rd European conference on The Semantic Web: research and applications (ESWC 2006)*, Springer-Verla. pages 411-426, 2006.

[18] S. Bao, G. Xue, X. Wu, Y. Yu, B. Fei, and Z. Su, Optimizing web search using social annotations, *In Proc. of WWW 2007*, pages 501âĂŞ510, 2007.

[19] D. Koop, C. E. Scheidegger, and S. P. Callahan, VisComplete: Automating Suggestions for Visualization Pipelines, *IEEE Transactions on Visualization and Computer Graphics*, vol.14, no. 6, pages 1691, 2008.

[20] E. Chinthaka, J. Ekanayake, and D, Leake CBR Based Workflow Composition Assistant, *In Proc. of 2009 World Congress on Services (SERVICES-I)*, IEEE Computer Society. pages 352-355.

[21] J. Zhang, W. Tan, and J. Alexander, Recommend-As-You-Go: A Novel Approach Supporting Services-Oriented Scientific Workflow Reuse, *In Proc. of the 2011 IEEE International Conference on Services Computing (SCC 2011)*, pages 48-55.

[22] J. Li, Z. Su, Z. Q. Ma, R. J. Slebos, P. Halvey, D. L. Tabb, D. C. Liebler, W. Pao, and B. Zhang, A bioinformatics workflow for variant peptide detection in shotgun proteomics, *Molecular & Cellular Proteomics* 10.5 (2011): M110-006536.

[23] Crasto, C. Joaqium, S. H. Koslow, and K. Fissell, Workflow-based approaches to neuroimaging analysis, *Neuroinformatics*, Humana Press, pages 235-266, 2007.

[24] W. Michener, J. Beach, S. Bowers, L. Downey, M. Jones, B. LudÃďscher, D. Pennington, A. Rajasekar, S. Romanello, M. Schildhauer, D. Vieglais, and J. Zhang, Data integration and workflow solutions for ecology, *Data integration in the life sciences*, Springer Berlin Heidelberg, pages 321-324, 2005.

[25] R.S. Barga, J. Jackson, and N. Araujoe, Trident: Scientific Workflow Workbench for Oceanography, *In Proc.of SERVICES I*, pages 465-466, 2008.

[26] G. Singh, M. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, and G. Mehta, Workflow task clustering for best effort systems with Pegasus, *In Proc.of the 15th ACM Mardi Gras conferences (MG 2008)*, pages 235-266.

[27] A. Dolgert, L. Gibbons, and C. D. Jones, Provenance in high-energy physics workflows, *Computing in Science & Engineering*, vol.10, no. 3, pages 22-29, 2008.

[28] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, and D. D. Roure, myExperiment: a repository and social network for the sharing of bioinformatics workflows, *Nucleic acids research*, 38(suppl 2), W667-W682.

[29] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, Pegasus: A framework for mapping complex scientific workflows onto distributed systems, *Scientific Programming Journal*, vol.13, no. 3, pages 219-237, 2005.

[30] X. Fei, S. Lu, A Collectional Data Model for Scientific Workflow Composition, *In Proc. of the 2010 IEEE International Conference on Web Services (ICWS 2010)*, pages 567-574.

[31] J. Wang, D. Crawl, and I. Altintas, Big data applications using workflows for data parallel computing, *Computing in Science & Engineering*, vol.16, no. 4, pages 11-21, 2014.

[32] X. Fei, S. Lu, and C. Lin, A MapReduce-Enabled Scientific Workflow Composition Framework, *In Proc. of the 2009 IEEE International Conference on Web Services (ICWS 2009)*, pages 663-670.

[33] T. McPhillips, S.Bowers, and B. LudÃď scher, Collection-oriented scientific workflows for integrating and analyzing biological data, *In Proc. of DILS*, vol. 4075, 2006, pages 248âĂŞ263.

[34] D. Turi, P. Missier, C. Goble, D. D. Roure, and T. Oinn, Taverna workflows: Syntax and semantics, *In Proc. of eScience*, 2007, pages 441âĂŞ448.

[35] S. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo, VisTrails: visualization meets data management, *In Proc. of SIGMOD*, 2006, pages 745âĂŞ747.

[36] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, Scaling Memcache at Facebook, *In Proc. of the 10th USENIX conference on Networked Systems Design and Implementation (NSDI 2013)*, pages 385-398.

[37] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, Bigtable: A distributed storage system for structured data, *In Proc. of the ACM Transactions on Computer Systems (TOCS 2008)*, vol. 26, no. 2 (2008).

[38] M.N. Vora, Hadoop-HBase for large-scale data, *In proc. of the 2011 International Conference on Computer Science and Network Technology (ICCSNT 2011)*, pages 601-605.

[39] R. Angels, C. Gutierrez, Survey of graph database models, *In proc of the ACM Computing Surveys (CSUR'08)*, vol. 40, no. 1 (2008), 1.

[40] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, *In Proc. of OSDI*, 2004, pages 137-150.

[41] C. Olston, B. Reed, and U. Srivastava Pig Latin: a not-so-foreign language for data processing, *In Proc. of SIGMOD*, 2008, pages 1099-1110.

[42] Y. Yu, M. Isard, D. Fetterly, M. Budiu, ÃŽ. Erlingsson, P. K. Gunda, and J. Currey, DryadLINQ: A system for generalpurpose distributed data-parallel computing using a high-level language, *In Proc. of OSDI*, 2008, pages 1-14.

[43] D. Williams, The Arbitron national in-car study, *Arbitron Inc.*, 2009.

[44] Chunhyeok Lim, S. Lu, A. Chebotko, F. Fotouhi, and A. Kashlev, OPQL: Querying Scientific Workflow Provenance at the Graph Level, *Data & Knowledge Engineering (DKE)*, 88(2013), pages 37-59, 2013.

[45] D. Ruan, S. Lu, A. Mohan, X. Fei, and J. Zhang, A User-Defined Exception Handling Framework in the VIEW Scientific Workflow Management System, *In Proc. of the IEEE International Conference on Services Computing (SCC)*, pages 274-281, Honolulu, Hawaii, 2012.

[46] G. Bell, T. Hey, and A. Szalay, Beyond the data deluge, *Science* 323.5919 (2009), pages 1297-1298.

[47] H. Chen, R. H. L. Chiang, and V. C. Storey, Business Intelligence and Analytics: From Big Data to Big Impact, *MIS quarterly*, 36.4 (2012), pages 1165-1188.

[48] J. Han, Haihong. E, Survey on NoSQL database, *In Proc. of the 2011 6th international conference on Pervasive computing and applications (ICPCA)*, 2011, pages 363-366.

[49] J. Pokorny, NoSQL databases: a step to database scalability in web environment, *International Journal of Web Information Systems*, 9.1 (2013), pages 69-82.

[50] A. Chebotko, A. Kashlev, and S. Lu, A Big Data Modeling Methodology for Apache Cassandra. *In Proc. of the 2015 IEEE International Congress on Big Data(BigData Congress)*, 2015, pages 238-245

[51] S. Abrishami, M.Naghibzadeh, and D. H. J. Epema, Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds. *Future Generation Computer Systems (FGCS)*, vol. 29, no. 1, pages 158-169, 2013.

[52] Z. Wu, Z. Ni, A revised discrete particle swarm optimization for cloud workflow scheduling. *In Proc. Of the International Conference Computational Intelligence and Security (CIS)*, pages 184-188, 2010.

[53] S. Yassa, R.Chelouah, H.Kadima, and B.Granado, Multi objective approach for energy-aware workflow scheduling in cloud computing environments. *The Scientific World Journal*, Volume 2013(2013), Article ID 350934.

[54] R. N. Calheiros, R. Buyya, Meeting deadlines of scientific workflows in public clouds with tasks replication. *IEEE Transaction Parallel and Distributed Systems*, vol. 25, no. 7, pages 1787âĂŞ1796, 2014.

[55] M. Malawski, G. Juve, Cost and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. *In Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Article No. 22, 2012.

[56] A. C. Zhou, B. He, Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds. *IEEE Transactions on Cloud Computing*, vol. 4, no. 1, pages 34-48, 2015.

[57] C. Lin, S. Lu, SCPOR: An elastic workflow scheduling algorithm for services computing. *In Proc. of the International Conference on Service Oriented Computing and Applications (SOCA 2011)*, pages 1-8, 2011.

[58] M.A. Rodriguez, R. Buyya, A responsive Knapsack-based algorithm for resource provi-sioning and scheduling of scientific workflows in clouds. *In Proc. of the 44th International Conference on Parallel Processing (ICPP 2015)*, pages 839-848, 2015.

[59] A. Mohan, S. Lu, and A. Kotov, Addressing the Shimming Problem in Big Data Scientific Workflows. *In Proc. of the 2014 IEEE International Conference on Services Computing (SCC 2014)*, pages 347-354, 2014.

[60] M. Xu, L. Cui, A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing. *In Proc. of the International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 629-634, 2009.

[61] T. T. Huu, J.Montagnat, Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. *In Proc. of the International Conference Cluster, Cloud Grid Computing (CCGrid)*, pages 612-617, 2010.

[62] D. de Oliveira, K.A. OcaÃśa, F.BaiÃčo, and M.Mattoso, A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *Journal of Grid Computing*, vol.10, no. 3, pages 521âĂŞ552, 2012.

[63] A. Kashlev, S. Lu, A System Architecture for Running Big Data Workflows in the Cloud. *In Proc. of the 2014 IEEE International Conference on Services Computing (SCC 2014)*, pages 51-58, 2014.

[64] A. Mohan, M. Ebrahimi, S.Lu, and A.Kotov, A NoSQL Data Model for Scalable Big Data Workflow Execution. *In Proc. of the International IEEE Congress on Big Data (BigData 2016)*, pages 52-59, 2016.

[65] M. Ebrahimi, A.Mohan, S.Lu, and R.Reynolds, TPS: A task placement strategy for big data workflows. *In Proc. of the 2015 IEEE International Conference on Big Data (Big Data)*, pages 523-530, 2015.

[66] M. Ebrahimi, A. Mohan, A. Kashlev, and S. Lu, BDAP: A Big Data Placement Strategy for Cloud-Based Scientific Workflows. *In Proc. of the 2015 IEEE First International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 105-114, 2015.

[67] P. Mell, T. Grance, The NIST definition of cloud computing. *Communications of the ACM*, vol. 53, no. 6, pages 50, 2010.

[68] A.Lenk, M.Menzel, J.Lipsky, S.Tai, and P.Offermann, What are you paying for? performance benchmarking for infrastructure-as-a-service offerings. *In Proc. of the 2011 IEEE International Conference on Cloud Computing (CLOUD)*, pages 484-491, 2011.

[69] A. Lenk, M.Klems, J.Nimis, S.Tai, and T.Sandholm, Whatś inside the Cloud? An architectural map of the Cloud landscape. *In Proc. of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23-31, 2009.

[70] H. Arabnejad, J.G. Barbosa, A budget constrained scheduling algorithm for workflow applications. *Journal of Grid Computing*, vol. 12, no. 4, pages 665-679, 2014.

[71] R. Sakellariou, H.Zhao, Scheduling workflows with budget constraints. *in Integrated research in GRID computing*, pages 189-202. Springer US, 2007.

[72] W. Zheng, R.Sakellariou, Budget-deadline constrained workflow planning for admission control. *Journal of Grid Computing*, vol. 11, no. 4, pages 633-651, 2013.

[73] H.Arabnejad, J.G. Barbosa, and R. Prodan, Low-time complexity budget-deadline constrained workflow scheduling on heterogeneous resources. *Future Generation Computer Systems (FGCS)*, vol. 55, pages 29-40, 2016.

[74] R. Prodan, M.Wieczorek, Bi-criteria scheduling of scientific grid workflows. *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 2, pages 364-376, 2010.

[75] J. Yu, R.Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, vol. 14, no. 3-4, pages 217-230, 2006.

[76] Vouk, M.A. and M.P. Singh, Quality of service and scientific workflows. Quality of Numerical Software, 1996. 76: pages 77-89.

[77] Ogasawara, E., et al., An algebraic approach for data-centric scientific workflows. Proc. of VLDB Endowment, 2011. 4(12): pages 1328-1339.

[78] Juve, G. and E. Deelman, Scientific workflows and clouds. Crossroads, 2010. 16(3): pages 14-18.

[79] Bharathi, S., et al. Characterization of scientific workflows. in Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on. 2008.

[80] Weiss, A., Computing in the clouds. networker, 2007. 11(4).

[81] Foster, I., et al. Cloud computing and grid computing 360-degree compared. in Grid Computing Environments Workshop, 2008. GCE 2008. 2008.

[82] Lohr, S., Google and IBM join in cloud computing research. New York Times, 2007. 8.

[83] Ricadela, A., Computing heads for the clouds. Business Week, 2007.

[84] Juve, G., et al., Characterizing and profiling scientific workflows. Future Generation Computer Systems, 2013. 29(3): pages 682-692.

[85] Brantner, M., et al. Building a database on S3. in Proceedings of the 2008 ACM SIGMOD international conference on Management of data. 2008. ACM. 13.

[86] Buyya, R., C.S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. in High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on. 2008.

[87] Moretti, C., et al. All-pairs: An abstraction for data-intensive cloud computing. in Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on. 2008.

[88] Zhao, Y., et al. Opportunities and challenges in running scientific workflows on the cloud. in Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 In

[89] ternational Conference on. 2011.

[90] Hoffa, C., et al. On the use of cloud computing for scientific workflows. in eScience, 2008. eScience'08. IEEE Fourth International Conference on. 2008.

[91] Ji Liu, E.P., Patrick Valduriez, Marta Mattoso, Parallelization of Scientific Workflows in the Cloud. 2014.

[92] Hey, A.J., S. Tansley, and K.M. Tolle, The fourth paradigm: data-intensive scientific discovery. Vol. 1. 2009: Microsoft Research Redmond, WA.

[93] Gilbert, S. and N. Lynch, Brewers conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, 2002. 33(2): pages 51-59.

[94] Birman, K., Q. Huang, and D. Freedman, Overcoming the 'D'in CAP: Using Isis2 to Build Locally Responsive Cloud Services. Computer, 2011: pages 50- 58.

[95] Agrawal, D., et al., Data management challenges in cloud computing infrastructures, in Databases in Networked Information Systems. 2010, Springer. pages 1-10.

[96] Stewart, R.J., P.W. Trinder, and H. W. Loidl, Comparing high level mapreduce query languages, in Advanced Parallel Processing Technologies. 2011, Springer. pages 58-72.

[97] Grossman, R. and Y. Gu. Data mining using high performance data clouds: experimental studies using sector and sphere. in Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. 2008. ACM.

[98] NIST Big Data Working Group. Available from: http://bigdatawg.nist.gov/.

[99] Brown, D.A., et al., A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis. 2006: Springer.

[100] Demchenko, Y., et al. Addressing big data issues in scientific data infrastructure. in Collaboration Technologies and Systems (CTS), 2013 International Conference on. 2013.

[101] Eaton, C., et al., Understanding big data: Analytics for enterprise class hadoop and streaming data. FREE ebook, 2012.

[102] Lotfy, A.E., et al., A middle layer solution to support ACID properties for NoSQL databases. Journal of King Saud University-Computer and Information Sciences, 2016. 28(1): pages 133-145.

[103] Lakshman, A. and P. Malik, Cassandra decentralized structured storage system. 2009. Cited on, 2015: pages 17.

[104] Chodorow, K., MongoDB: the definitive guide. 2013: " O'Reilly Media, Inc.".

[105] Abouelhoda, M., S.A. Issa, and M. Ghanem, Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. BMC bioinformatics, 2012. 13(1): pages 77.

[106] Emeakaroha, V.C., et al., Managing and optimizing bioinformatics workflows for data analysis in clouds. Journal of grid computing, 2013. 11(3): pages 407-428.

[107] Vockler, J.S., et al. Experiences using cloud computing for a scientific workflow application. in Proceedings of the 2nd international workshop on Scientific cloud computing. 2011. ACM.

[108] Wu, Z., et al., A market-oriented hierarchical scheduling strategy in cloud workflow systems. The Journal of Supercomputing, 2013. 63(1): pages 256-293.

[109] De Oliveira, D., et al. Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. in Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on. 2010.

[110] Durillo, J.J., V. Nae, and R. Prodan. Multi-objective workflow scheduling: An analysis of the energy efficiency and makespan tradeoff. in Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on. 2013.

[111] Tangudu, V. and M. Mishra. Estimating makespan using double trust thresholds for workflow applications. in Proceedings of the CUBE International Information Technology Conference. 2012. ACM.

[112] Weise, T., Global optimization algorithms-theory and application. Self- Published, 2009.

[113] Venugopal, S. and R. Buyya, An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids. Journal of Parallel and Distributed Computing, 2008. 68(4): pages 471-487.

[114] Deelman, E., et al., Pegasus: Mapping large-scale workflows to distributed resources. 2006: Springer.

[115] Google App Engine: Platform as a Service. Available from: https://cloud.google.com/appengine/docs.

[116] Microsoft Azure: Cloud Computing Platform & Services. Available from: http://azure.microsoft.com/en-us/.

[117] OpenStack. Available from: https://www.openstack.org/.

[118] Amazon Elastic Compute Cloud (EC2). Available from: http://aws.amazon.com/ec2/.

[119] FutureSystems: Digital Science Center, School of Informatics and Computing, Indiana University | FutureSystems Portal. Available from: https://portal.futuresystems.org/.

[120] Pegasus | Workflow Management System. Available from:
http://pegasus.isi.edu/index.php.

ABSTRACT

IMPROVING USABILITY AND SCALABILITY OF BIG DATA WORKFLOWS IN THE CLOUD.

by

Aravind Mohan

August 2017

Advisors: Dr. Shiyong Lu, Dr. Song Jiang

Major: Computer Engineering

Degree: Doctor of Philosophy

Big data workflows have recently emerged as the next generation of data-centric workflow technologies to address the five "V" challenges of big data: volume, variety, velocity, veracity, and value. More formally, a big data workflow is the computerized modeling and automation of a process consisting of a set of computational tasks and their data interdependencies to process and analyze data of ever increasing in scale, complexity, and rate of acquisition. The convergence of big data and workflows creates new challenges in workflow community.

First, the variety of big data results in a need for integrating large number of remote Web services and other heterogeneous task components that can consume and produce data in various formats and models into a uniform and interoperable workflow. Existing approaches fall short in addressing the so-called shimming problem only in an adhoc manner and unable to provide a generic solution. We automatically insert a piece of code called shims or adaptors in order to resolve the data type mismatches.

Second, the volume of big data results in a large number of datasets that needs to be queried and analyzed in an effective and personalized manner. Further, there is also a strong need for sharing, reusing, and repurposing existing tasks and workflows across different users and institutes. To overcome such limitations, we propose a folksonomy- based social workflow recommendation system to improve workflow design productivity and efficient dataset querying and analyzing.

Third, the volume of big data results in the need to process and analyze data of ever increasing in scale, complexity, and rate of acquisition. But a scalable distributed data model is

still missing that abstracts and automates data distribution, parallelism, and scalable processing. We propose a NoSQL collectional data model that addresses this limitation.

Finally, the volume of big data combined with the unbound resource leasing capability foreseen in the cloud, facilitates data scientists to wring actionable insights from the data in a time and cost efficient manner. We propose BARENTS scheduler that supports high-performance workflow scheduling in a heterogeneous cloud-computing environment with a single objective to minimize the workflow makespan under a user provided budget constraint.

AUTOBIOGRAPHICAL STATEMENT

Aravind Mohan is currently a PhD candidate in the Big Data Research Lab at Wayne State university under the supervision of Dr. Shiyong Lu. Before that, he worked in the industry as a software engineer. His research focuses on big data management and cloud computing. His broader areas of interests are services computing, online education services and information retrieval. He has published several research articles in peer-reviewed international conferences, including IEEE conference on services computing, big data congress, big data, big data computing services and applications and the ACM SIGIR conference. He is a member of IEEE and ACM. He has joined as a tenure track assistant professor at the department of comuter science in Allegheny College.